National Information Exchange Model — Model Package Description Specification

Version 3.0beta1

April 30, 2014

NIEM Technical Architecture Committee (NTAC)

Contents

Abstract

This document specifies normative rules and non-normative guidance for building Model Package Descriptions (MPDs) that conform to the National Information Exchange Model (NIEM) version 3.0.

Status

This document is a draft of the specification for NIEM Model Package Descriptions (MPDs). It represents the design that has evolved from the collaborative work of the NIEM Business Architecture Committee (NBAC) and the NIEM Technical Architecture Committee (NTAC) and their predecessors.

This specification is a product of the NIEM Program Management Office (PMO).

Email comments on this specification to `niem-comments@lists.gatech.edu`.

Major changes:

Alpha 11

1. Moved Section 4.7, *Conformance Assertion*, below, section to mandatory artifacts.
2. Added NIEM subset that supplements `mpd-catalog-3.0.xsd`.
3. Inserted proxy types (and namespace) into `mpd-catalog-3.0.xsd`.
4. Renamed `master-document` to `read-me.*`
5. Revised Appendix F, *Guidance for IEPD Directories (non-normative)*, below, per feedback and discussions.
6. Validated CoT and all other simpler IEP samples.
7. Provided initial cut `niem-core` subset for `mpd-catalog-3.0.xsd`.

Beta 1

1. Added Rule 4-9, below, and text above it.
2. Removed first two sentences of first paragraph of Section 7.3, *Artifact Links to Other Resources*, below.
3. Changed Rule 7-9, below, from MUST to SHOULD.
4. Lots of minor corrections to Appendix F, *Guidance for IEPD Directories (non-normative)*, below.
5. Revised Section 5, *Conformance Targets*, below.
6. Added Section 7.4, *IEPD Completeness*, below.
7. Added definition for ·XML schema assembly·.
8. Added statement that the MPD method for ·XML schema assembly· is use of XML catalogs.
9. Added Appendix for Appendix D, *Common MPD Artifacts*, below.
10. Revised wording of Rule 7-8, below, for Schematron.
11. Added "derives_from" to mpd-catalog-3.0.xsd to support revision to Rule 7-8, below.
12. Added EXI subdirectory to non-normative guidance.
13. Fixed Rule 4-11, below: only `c:mpdURI` attributes must adhere to absolute-URI production.

14. Added simple example of informal conformance assertion. Appendix E, *Conformance Assertion Example*, below
15. Added example MPD catalog document from Cursor on Target (Dr Renner). Appendix B, *Example MPD Catalog Document for Cursor on Target*, below
16. Added ref to MPD Toolkit that will contain entire CoT IEPD.
17. Added 3 Schematron rule files for a few conformance targets; declared them "alpha" and untested. Appendix C, *Schematron Rules for MPDs*, below
18. Assembled MPD Toolkit; initial version available at `http://reference.niem.gov/niem/resource/mpd/3.0/`; includes: mpd-spec-3.0beta1.pdf, mpd-catalog-3.0beta1.xsd, mpd catalog NIEM subset, Schematron rules (alpha), example IEPD for CoT, example conformance-assertion (template).

Remaining work and issues

1. Section 5 still needs more text and organization.
2. Add short title for each rule (approx 67 total).
3. Conformance targets (CT): missing? fork? class=Constraint/Interpretation? applicability=(correct CT)?
4. oXygen validation errors on MPD catalog XSD; add lower level cardinality constraints back in to resolve oXygen errors?
5. Remaining TBDs.
6. Final QA and validity checks on MPD catalog schema, other appendices, etc.
7. Final checks for typos, spell, punctuation, acronyms/abbrevs, missing refs or termRefs, consistency.

1. Introduction

This specification assumes familiarity with the National Information Exchange Model (NIEM), its basic concepts, architecture, processes, design rules, and general conformance rules. For novices to NIEM, the recommended reading list includes:

- Introduction to the National Information Exchange Model **[NIEM Introduction]**
- NIEM Concept of Operations **[NIEM Concept of Operations]**
- NIEM Naming and Design Rules **[NIEM NDR]**
- NIEM High-Level Version Architecture **[NIEM High-Level Version Architecture]**
- NIEM High-Level Tool Architecture **[NIEM High-Level Tool Architecture]**
- NIEM Conformance **[NIEM Conformance]**
- NIEM User Guide **[NIEM User Guide]**
- NIEM Business Information Exchange Components **[NIEM BIEC]**
- NIEM Implementation Guidelines **[NIEM Implementation Guidance]**
- NIEM Conformance Target Attribute Specification **[NIEM Conformance Target Attribute Specification]**

The foregoing NIEM documents are available at `http://reference.niem.gov/niem/`. See **[NIEM Implementation Guidance]** for NIEM Implementation Guidelines. Note that a few of these documents are relatively old, in particular the Concept of Operations and the User Guide. These documents are listed because they still provide a great deal of useful information for understanding NIEM. Just realize that some of the techniques and processes in these documents have changed since they were drafted.

Those knowledgeable of NIEM should be familiar with the **[NIEM NDR]**, **[NIEM High-Level Version Architecture]**, **[NIEM Conformance]**, **[NIEM Conformance Target Attribute Specification]**, and **[NIEM BIEC]**.

This MPD Specification v3.0 uses and is a peer to the NIEM Naming and Design Rules (NDR) **[NIEM NDR]**. It supersedes IEPD guidance previously published in Requirements for a NIEM IEPD **[NIEM IEPD**

Requirements] and the NIEM User Guide **[NIEM User Guide]**. The NIEM User Guide remains a good source for understanding the process of building Information Exchange Package Documentation (IEPD). It also supersedes both MPD Specification v1.0 and v1.1.

1.1. Background

Many fundamental concepts, processes, and products in the NIEM generally involve aggregating electronic files into logical sets that serve a specific purpose. Examples of such sets include, but in the future may not necessarily be limited to, a NIEM release, core update (CU), domain update (DU), Information Exchange Package Documentation (IEPD), and Enterprise Information Exchange Model (EIEM). Each of these examples is a NIEM Model Package Description (MPD).

[Definition: Model Package Description (MPD)]

A set of related W3C XML Schema documents and other supporting files organized as one of the five classes of NIEM schema sets:

- Release (major, minor, or micro).
- Domain update (DU) to a release.
- Core update (CU) to a release.
- Information Exchange Package Documentation (IEPD).
- Enterprise Information Exchange Model (EIEM).

An MPD is a fundamental NIEM artifact that represents a reusable or implementable NIEM-conformant information model. An MPD is packaged as a **[PKZIP]** archive file.

A key NIEM concept used throughout this specification is *data component*.

[Definition: data component]

An XML Schema type or attribute group definition; or an XML Schema element or attribute declaration.

An MPD is a normative specification for XML data components in the format of the World Wide Web Consortium (W3C) XML Schema Definition Language **[W3C XML Schema Datatypes]**, **[W3C XML Schema Structures]**. MPD schema documents either (1) define the semantics and structure for NIEM reusable data components, or (2) define implementable NIEM exchange instance documents in W3C Extensible Markup Language (XML) **[W3-XML]**.

An MPD is ready to publish and use when it conforms to NIEM specifications, and has been properly packaged with the schemas, documentation, and supplemental files needed to implement or reuse it. MPD content design, development, and assembly may be difficult and time-consuming, especially if done manually. Software tools can significantly reduce the complexity of designing, constructing, changing, and managing MPDs. In order to reduce ambiguity and to facilitate interoperable and effective tool support, this baseline specification imposes some degree of consistency on the terminology, syntax, semantics, and composition of MPDs.

1.2. Purpose

This document is a normative specification for the various kinds of NIEM MPDs. The rules and guidance herein are designed to encourage and facilitate NIEM use and tools by balancing consistency, simplicity, and flexibility. Consistency and simplicity make MPDs easy to design correctly, build rapidly, and find easily (for reuse or

adaptation). Consistency also facilitates tool support. Flexibility enables more latitude to design and tailor MPDs for complex data exchange requirements. As such, this document does not necessarily prescribe mandates or rules for all possible situations or organizational needs. If an organization determines it should impose additional constraints or requirements on its IEPDs beyond those specified in this document (for example, mandating a normative set of business requirements or a domain model within IEPD documentation), then it is free to do so, as long as no conflicts exist with this MPD Specification or the **[NIEM NDR]**.

This document defines terminology; identifies required and optional (but common) artifacts; defines metadata; specifies normative constraints, schemes, syntax, and processes as rules; provides non-normative guidance; and as needed, refers to other related NIEM specifications for more detail.

1.3. Scope

This specification applies to information exchange definitions and release products that employ the data component definitions and declarations in NIEM Core and Domains. It also applies to the NIEM release products and their associated updates. In particular, this version of this document applies to the following MPDs:

- NIEM releases (including major, minor, and micro releases).
- NIEM domain updates (DU) **[NIEM Domain Update Specification]**. (Note these are NOT the same as the NIEM domain schemas that are part of numbered releases).
- Core updates (CU) to NIEM releases.
- Information Exchange Package Documentation (IEPD) that define NIEM data exchanges.
- Enterprise Information Exchange Model (EIEM) from which one or more NIEM IEPDs can be built or based.

In the future, as required, other types of MPDs may be added to this list.

At any point in time, an incomplete MPD will be in some state of development. This specification is applicable to such developing products in that it establishes validity standards for MPDs in progress, as well as completeness standards for MPDs that reach a final, published, production-quality state. In turn, tool vendors should be able to build, adapt, and/or integrate software tools that will assist in MPD development and assembly from raw parts to finished product.

NIEM is a data layer for an information architecture. Files in an MPD generally define XML Schema types and declare XML elements and attributes to use in payloads for information exchanges. While an MPD may also contain files from layers beyond the data layer, this specification is not intended to define details of other architectural layers. Such files are generally present only to provide additional context, understanding, or assistance for implementing the exchange of payloads.

Authoritative sources are not required to revise MPDs that exist before this specification becomes effective. However, they are always encouraged to consider revising MPDs to meet this specification, especially when making other significant changes.

1.4. Audience

The following groups should review and adhere to this specification:

- The NIEM release manager who is responsible to integrate and publish NIEM releases and core updates.
- NIEM domain stewards and technical representatives who develop and publish domain updates.
- NIEM IEPD developers and implementers.
- NIEM-aware tool developers and vendors.
- Organizations that intend to develop an EIEM.
- Individuals or groups responsible to review and approve MPDs.

2. Basic Concepts and Terminology

The presentation of concepts and terms in this section is sequenced for understanding. Each subsection builds upon previous ones. This section concludes with an explanation of each of the five MPD classes and a summary of their similarities and differences.

2.1. Key Words for Requirement Levels

Within normative content rules and definitions, the key words MUST, MUST NOT, SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, RECOMMENDED, REQUIRED, and OPTIONAL in this document are to be interpreted as described in **[RFC 2119 Key Words]**.

2.2. Character Case Sensitivity

This specification imposes many constraints on the syntax for identifiers, names, labels, strings, etc. In all cases, unless otherwise explicitly noted, syntax is case sensitive. In particular, XML files in appendices that define particular artifacts, transformations, and examples are case sensitive.

Also, note that as a general principle, lower case characters are used whenever such will not conflict with the **[NIEM NDR]**.

2.3. Artifacts

MPDs are generally composed of files and file sets grouped for a particular purpose. Each file is referred to as an *artifact*, and each logical set of such files is called an *artifact set*.

[Definition: artifact]

A single file with a defined purpose.

[Definition: artifact set]

A collection of ·artifacts· logically grouped for a defined purpose.

An MPD is itself a set of artifacts, the purpose for which is to define and document the intended use of the MPD. While the key MPD artifacts are its ·XML schema document· artifacts, there are also other kinds of MPD artifacts. These may include (but are not limited to) HTML, XSLT, text, or graphic files used for human-readable documentation. An MPD may also have artifacts intended to help assist in or accelerate the use and implementation of the MPD. For example, these may be XML, UML, or binary files that are inputs to or outputs from software tools used to build, generate, or edit the MPD or its schema document artifacts. Appendix D, *Common MPD Artifacts*, below, contains a listing of mandatory and common optional artifacts for the five types of MPDs. Common types of artifacts are described in more detail in subsequent sections. Section 7.1, *Artifact Sets*, below, discusses the different methods for grouping MPD artifacts into sets.

2.4. Schema Document and Namespace Correspondence in NIEM

To simplify automatic schema processing and reduce the potential for confusion and error, **[NIEM NDR]** principles state that each NIEM-conformant namespace SHOULD be defined by exactly one reference or extension schema document. To support this concept, the **[NIEM NDR]** disallows the use of `xs:include`, and mandates the use of the `xs:schema/@targetNamespace` attribute in NIEM-conformant schema documents.

So, (1) each NIEM namespace is defined by a single NIEM-conformant schema document, and (2) each NIEM-conformant schema document declares a target namespace. NIEM does not permit schema documents without target namespaces, unless they are from sources outside of NIEM.

2.5. Harmonization

Harmonization is a process that NIEM governance committees and domain stewards iteratively apply to NIEM content (specifically, its semantics, structure, and relationships) during the preparation of a NIEM major or minor release. On a more restricted scale a domain steward harmonizes his/her own content (schema documents) in preparation for a ·domain update· MPD. Multiple domain stewards may collaborate in a coordinated ·domain update·. In this case, to the extent possible, harmonization may be applied across the content of all the collaborating domains. Harmonization results in model change and evolution with the intent of removing semantic duplication and overlap while improving representational quality and usability.

[Definition: harmonization]

Given a data model, harmonization is the process of reviewing its existing data definitions and declarations; reviewing how it structures and represents data; integrating new data components; and refactoring data components as necessary to remove (or reduce to the maximum extent) semantic duplication and/or semantic overlap among all data structures and definitions resulting in quality improvements to representation and use.

2.6. XML Validation

A discussion of XML validation requires a basic understanding of basic XML terminology. The following definitions are necessary.

[Definition: XML document]

A document in XML format as defined by **[W3-XML]**, §2.

[Definition: schema component]

A *schema component* is as defined by **[W3C XML Schema Structures]** §2.2, "XML Schema Abstract Data Model", which states:

> **Schema component** is the generic term for the building blocks that comprise the abstract data model of the schema.

[Definition: XML Schema]

An *XML Schema* is as defined by **[W3C XML Schema Structures]** §2.2, "XML Schema Abstract Data Model", which states:

> An **XML Schema** is a set of schema components.

[Definition: XML schema validation]

The process of checking an ·XML document· to confirm that it is both *well-formed* **[W3-XML]**, §2.1 Well-Formed XML Documents and *valid* **[W3C XML Schema Structures]**, §2.3 Constraints and Validation Rules, in that it follows the structure defined by an associated ·XML Schema·. A well-formed document follows the basic syntactic rules of XML, which are the same for all XML documents.

[Definition: XML schema document]

A physical (file) representation of part or all of an ·XML Schema·. One or more *XML schema documents* are used to assemble ·schema components· into an ·XML Schema·.

[Definition: XML schema assembly]

A process that uses ·XML schema documents· to identify the constituent ·schema components· for an ·XML Schema·, and correctly sequences and structures these components to construct a single entity, the ·XML Schema·.

In other words, an ·XML Schema· is the result of ·XML schema assembly·, processing a set of one or more ·XML schema documents· into a single entity. That entity is most commonly an electronic image in the memory of a computer.

This specification often refers to the process of ·XML schema validation·, that is, validation of an instance XML document to confirm it adheres to the structure defined by a particular ·XML Schema·. Generally, this should occur periodically during and after design time to ensure the conformance and quality of an information exchange definition (i.e., ·XML schema documents·) and associated instance XML documents. However, local architecture or policy may dictate the need to validate more often, and in some cases may require runtime validation.

XML schema document sets that define a NIEM information exchange must be authoritative. Application developers may use other schemas (e.g., constraint or Schematron schema documents) for various purposes, but for the purposes of determining NIEM conformance, the authoritative reference schema documents (NIEM releases) are relevant. This does not mean that XML validation must be performed on all instance XML documents as they are served or consumed; only that the instance XML documents validate if and when XML validation is performed. Therefore, even when validation is not performed, instance XML documents must be valid against the XML schema that is assembled from XML schema document sets that specify these instance XML documents.

2.7. Reference Schema Documents

A NIEM *reference schema document* is a schema document that is intended to be the authoritative definition of business semantics for components within its target namespace. Reference schema documents include the NIEM Core schema documents, NIEM domain schema documents, and NIEM domain update schema documents. The normative definition for a reference schema document and applicable conformance rules are found in the **[NIEM NDR]**. The definition is repeated here:

[Definition: reference schema document]

An XML Schema document that meets all of the following criteria:

- It is a conformant schema document.
- It is explicitly designated as a reference schema document. This may be declared by an MPD catalog or by a tool-specific mechanism outside the schema document.
- It provides the broadest, most fundamental definitions of components in its namespace.
- It provides the authoritative definition of business semantics for components in its namespace.
- It is intended to serve as the basis for components in IEPD and EIEM schema documents, including subset, constraint, and extension schema documents.

See also ·reference schema document set·.

[Definition: reference schema document set]

A set of related reference schema documents, such as a NIEM release. See also ·reference schema document·.

The **[NIEM NDR]** conformance rules for reference schema documents are generally stricter than those for other classes of NIEM-conformant schema documents. For example, they are not allowed to employ particular XML Schema model groups such as `xs:choice` or `xs:any` that other schema documents may contain.

NIEM reference schemas are very uniform in their structure. As they are the primary definitions for data components, they do not need to restrict other data definitions, and they are not allowed to use XML Schema's complex type restriction mechanisms.

2.8. Rules

Rules state specific requirements on artifacts or on the interpretation of artifacts. The classes of artifacts are identified by ·conformance targets· that are enumerated by this document in Section 2.9, *Conformance Target Concepts*, below. Rules are normative.

[Rule <section>-<number>] (<applicability>) (<class>)

An enforceable rule for NIEM.

Each rule has a classification, which is either *Constraint* or *Interpretation*. If the classification is *Constraint*, then the rule is a ·constraint rule·. If the classification is *Interpretation*, then the rule is an ·interpretation rule·.

[Definition: constraint rule]

A rule that sets a requirement on an artifact with respect to its conformance to a ·conformance target·.

[Definition: interpretation rule]

A rule that sets the methodology, pattern, or procedure for understanding or using some aspect of an instance of a conformance target.

Each rule has a list of one or more ·conformance targets· to which it applies. Each entry in the list is a code from

Table 2-1, *Conformance Target Codes*, below. Each code in the applicability list for a rule indicates that the rule applies to the corresponding conformance target. The conformance targets for this specification are summarized and coded in Section 2.9, *Conformance Target Concepts*, below. Section 5, *Conformance Targets*, below, provides additional discussion and rules for conformance targets.

Rules are numbered according to the section in which they appear and the order in which they appear within that section. For example, Rule 4-1 is the first rule in Section 4.

2.9. Conformance Target Concepts

This section introduces the concept of ·conformance target·, a concept fundamental to understanding the normative rules defined in this specification. Section 5, *Conformance Targets*, below, defines and discusses rules for specific ·conformance targets· established for MPDs.

2.9.1. Conformance Targets Attribute Specification terminology

The **[NIEM Conformance Target Attribute Specification]** defines several terms used normatively within this specification.

[Definition: conformance target]

The term *conformance target* is as defined by **[NIEM Conformance Target Attribute Specification]**, which states:

> A *conformance target* is a class of artifact, such as an interface, protocol, document, platform, process or service, that is the subject of conformance clauses and normative statements. There may be several conformance targets defined within a specification, and these targets may be diverse so as to reflect different aspects of a specification. For example, a protocol message and a protocol engine may be different conformance targets.

[Definition: conformance target identifier]

The term *conformance target identifier* is as defined by **[NIEM Conformance Target Attribute Specification]**, which states:

> A *conformance target identifier* is an internationalized resource identifier that uniquely identifies a conformance target.

[Definition: effective conformance target identifier]

The term *effective conformance target identifier* is as defined by **[NIEM Conformance Target Attribute Specification]**, which states:

> An *effective conformance target identifier* of a conformant document is an internationalized resource identifier reference that occurs in the document's effective conformance targets attribute.

2.9.2. Conformance Targets for MPDs

There are numerous use cases for MPDs, and in particular, IEPDs. These use cases, and the rules associated with each, direct the list of conformance targets specified by this MPD Specification. From a conformance target perspective, releases, Core and Domain updates, IEPDs, and EIEMs are special kinds of MPDs, and this establishes an *IS-A* relationship.

There are several purposes for defining conformance targets in NIEM specifications. A conformance target establishes a class of artifact. A class of artifact identified by a conformance target is associated with a set of rules. Based on these rules, tools and operations may be developed. However, a conformance target also provides purposes outside the environment of rules and tools.

Conformance targets satisfy a need to ensure developers do not conform to NIEM in name only. Once committed to using NIEM, developers and organizations need well-defined conformance targets and rules to know exactly how to conform to NIEM. Funding agencies require conformance targets that correspond to interoperability goals. An agency funding development of a set of systems will need to ensure it funds development of NIEM-conformant IEPDs that support the exchange of NIEM-conformant IEPs. Tools and system developers need conformance targets that identify real world requirements corresponding to their use cases and tool capabilities. Many of these tools are not yet developed. And so, this specification attempts to cover a broad range of general use cases.

Section 5, *Conformance Targets*, below, defines MPD Specification conformance targets and associated rules in more detail. However, the following is a brief summary of conformance targets:

- **Well-formed MPD** — This satisfies the need for a **[PKZIP]** file with an MPD catalog which follows a base set of rules: MPD catalog links to XML catalogs and identifies NDR classes of schema documents. IEPDs, EIEMs, releases, and their associated core and domains updates will conform to (e.g., *is-a*) this conformance target.
- **Full NIEM IEP** — This is an XML document with an XML document element that is defined by a NIEM reference schema or NIEM extension schema.
- **A NIEM IEPD** — This is a special kind of MPD that establishes at least one conformance target for a class of IEPs (by employing one or more `c:IEPConformanceTarget` elements).
- **IEP** — This is an XML document that conforms to a conformance target defined by a NIEM IEPD.
- Other conformance targets internal to the MPD specification, including **MPD catalog** and **XML catalog**.

The conformance targets above support a variety of use cases:

- An XML document that has a NIEM-defined XML document (root) element.
- An XML document that has a NIEM-defined document wrapped in a non-NIEM element (e.g. an envelope, which includes SOAP, LEXS, ULEX, Trusted Data Format, OGC Web Services, etc.).
- Use of multiple NIEM-defined payloads together, in an envelope, or as a composite package.
- Use of NIEM-defined data annotating and providing details to a non-NIEM format, such as OGC.
- Use of a non-NIEM format, where NIEM annotations and details may or may not occur depending on usage (e.g., NIEM details are provided for hospitals, but this message does not have any hospitals within it).
- The use of non-NIEM data annotating and providing details on NIEM data.

Table 2-1: Conformance Target Codes

Conformance Target	Code
Well-formed MPD	WF-MPD
Full NIEM IEP	FN-IEP
NIEM IEPD	N-IEPD
IEP	IEP

MPD catalog	M-Cat
MPD changelog	M-Chg-Log
XML catalog	X-Cat

[TBD -- What other conformance targets are needed?]

2.10. MPD Types

This section details the five classes of MPDs currently defined in NIEM.

2.10.1. NIEM Release

A NIEM *release* is an MPD containing a full set of harmonized reference schema documents that define and declare all content within a single version of NIEM. NIEM releases include major, minor, and micro releases (as defined in the **[NIEM High-Level Version Architecture]**).

[Definition: release]

A reference schema document set published by the NIEM Program Management Office (PMO) at `http://release.niem.gov/` and assigned a unique version number. Each schema document in the set defines data components for use in NIEM information exchanges. Each release is independent of other releases, although a schema document may occur in multiple releases. A release is of high quality, and has been vetted by NIEM governance bodies. A numbered release may be a major, minor, or micro release.

Current real examples of NIEM releases include NIEM major releases 1.0, 2.0, and 3.0, and minor release 2.1. Each numbered release is a reference schema document set that includes a NIEM Core (along with the various infrastructure and code list schema documents that supplement Core) and NIEM domain schema documents.

[Definition: major release]

A NIEM release in which the NIEM Core reference schema document has changed since previous releases. The first integer of the version number indicates the major release series; for example, versions 1.0, 2.0, and 3.0 are different major releases.

[Definition: minor release]

A NIEM release in which the NIEM Core has not changed from previous releases in the series, but at least one or more domain reference schema documents have changed. A second digit greater than zero in the version number indicates a minor release (for example, v2.1). Note also that major v2.0 and minor v2.1 are in the same series (i.e., series 2) and contain the same NIEM Core schema document.

[Definition: micro release]

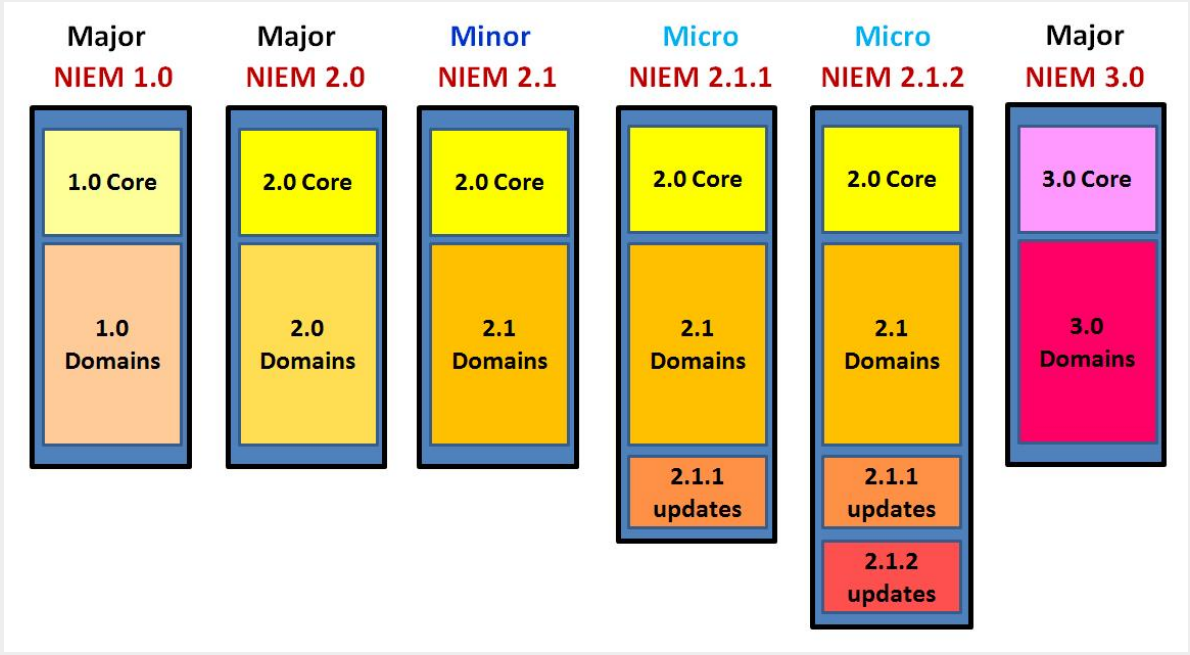
A NIEM release in which neither the NIEM Core nor the domain reference schema documents have changed from the previous major or minor release, but one or more new reference schema documents have been added (without impact to domain or Core schemas). A third digit greater than zero in the

version number indicates a micro release (for example, v2.1.1; note that this release does not currently exist).

A micro release is a NIEM release that adds new data components to the Core, domains, or both without removing or modifying existing Core and domain schemas or content. Figure 2-1, *Examples of NIEM numbered releases*, below, illustrates both real (v1.0, v2.0, v2.1, and v3.0) and fictitious (v2.1.1 and v2.1.2) examples of major, minor, and micro release composition.

Note that a given NIEM reference schema document (target namespace) can exist in multiple numbered releases. For example, as illustrated in Figure 2-1, *Examples of NIEM numbered releases*, below, both NIEM 2.0 and 2.1 contain (and reuse) the same NIEM Core 2.0 schema document. The **[NIEM High-Level Version Architecture]** defines the processes for numbering releases and identifying the schema documents that compose these sets. This specification outlines a similar version numbering scheme for MPDs and their artifacts Section 4.2.1, *Version Numbering Scheme*, below.

Figure 2-1: Examples of NIEM numbered releases



2.10.2. Domain Update

A *domain update (DU)* is an MPD containing a reference schema document or document set and a change log that represent changes to NIEM domains. The **[NIEM High-Level Version Architecture]** defines a domain update as both a process and a NIEM product. Through use and analysis of NIEM releases and published content, domain users will identify issues and new data requirements for domains and sometimes Core. NIEM domains use these issues as the basis for incremental improvements, extensions, and proposed changes to future NIEM releases. Both the process and product of the process are referred to as domain update. This MPD Specification is applicable to a domain update product.

[Definition: domain update]

An MPD that contains a reference schema document or document set issued by one or more domains that constitutes new content or an update to content that was previously published in a NIEM release. Domain updates are published to the NIEM Publication Area at

`http://publication.niem.gov/niem/` and available for immediate use within IEPDs.

A domain update may define and declare new versions of content applicable to a NIEM release or other published content. The issuing domain or domains vet each update, but the update is not subject to review by other NIEM governance. Before publication, domain updates are technically reviewed for and must satisfy NIEM-conformance, but otherwise have fewer constraints on quality than do NIEM releases.

A domain update may apply to one or more domain namespaces within a single NIEM major, minor, or micro release. A domain steward uses a domain update to: (1) make new or changed domain content immediately available to NIEM data exchange developers between NIEM releases, and (2) request that new or changed content be harmonized into a future NIEM release. (See **[NIEM Domain Update Specification]** which provides normative details about domain updates and the associated processes.)

2.10.3. Core Update

When necessary, the NIEM PMO can publish a *core update (CU)*. This is essentially identical to a domain update in terms of structure and use, with two important exceptions. First, a core update records changes that apply to a particular NIEM core version or another core update. This also means it is applicable to all NIEM releases using that same core version. Second, a core update is never published to replace a NIEM core. It is intended to add new schemas, new data components, new code values, etc. to a core without waiting for the next major release. In some cases, minor modifications to existing data components are possible.

[Definition: core update]

An MPD that applies changes to a given NIEM core schema document or document set. A core update never replaces a NIEM core; instead, it is used to add new schema documents, new data components, new code values, etc. to a particular NIEM core. In some cases, a core update can make minor modifications to existing core data components.

As with domain updates, all core updates are published to the NIEM Publications Area, their changes are immediately available for use in IEPDs, and they will be harmonized and integrated into the next major NIEM release.

2.10.4. Information Exchange Package Documentation (IEPD)

NIEM *Information Exchange Package Documentation (IEPD)* is an MPD that defines a class of instance XML documents that represent a recurring XML data exchange.

[Definition: Information Exchange Package Documentation (IEPD)]

An MPD that defines one or more (generally recurring) XML data or information exchanges.

A NIEM IEPD contains a NIEM-conformant XML schema document set that may include portions of a NIEM Core schema document (and updates), portions of NIEM Domain schema documents (and updates), enterprise-specific or IEPD-specific extension schema documents, and declares at least one ·IEP conformance target· within its MPD Catalog Section 4.1, *NIEM MPD Catalog*, below. The XML schema documents contained in an IEPD work together to define one or more classes of instance XML documents that consistently encapsulate data for meaningful information exchanges. Furthermore, any instance XML document that is valid for an XML schema document set (in the IEPD) and an associated ·IEP conformance target· (declared in the IEPD) is considered a member of that ·IEP conformance target· class. XML schema documents in a NIEM IEPD

conform to the **[NIEM NDR]** and may use or extend data component definitions drawn from NIEM. An IEPD may also incorporate and use XML schema documents from other standards that do not conform to NIEM. (See **[NIEM NDR]** for details.)

An IEPD consists of a set of artifacts (XML schema documents, documentation, sample instance XML documents, etc.) that together define and describe an implementable NIEM information exchange. An IEPD should contain an XML schema document set and instructional material necessary to:

- Understand information exchange context, content, semantics, and structure.
- Create and validate XML documents defined by the IEPD, and used for information exchanges.
- Identify the lineage of the IEPD itself and optionally its artifacts.

A NIEM IEPD defines one or more classes of XML documents. Each of these XML documents is an *Information Exchange Package (IEP)* that satisfies all validity constraints for its class as defined by the IEPD. An IEP is an information message payload serialized as XML and transmitted in some way, for example over a communications network. (**[FEA Data Reference Model]** and **[GJXDM IEPD Guidelines]** are the original sources of the terms *information exchange package* and *information exchange package documentation*, respectively).

> **[Definition: Information Exchange Package (IEP)]**
>
> An XML document that satisfies all the validity constraints for its class as defined by a NIEM IEPD.

How to declare validity constraints for one or more IEP classes within an IEPD will be covered in more depth in Section 4.6, *Information Exchange Packages*, below.

Note that NIEM conformance does not require that an IEP be native XML on the transmission medium. A NIEM-conformant IEP may be encrypted, compressed (e.g., using **[PKZIP]**, **[RAR]**, **[W3-EXI]**, etc.), or wrapped within an envelope mechanism, as long as its original native XML form can be retrieved by the receiver.

2.10.5. Enterprise Information Exchange Model (EIEM)

As an organization develops IEPDs, it may realize that many of its IEPDs have similar business content. A collection of closely related business data could be organized at an object level and defined as extension data components. In NIEM, these extension components are referred to as *Business Information Exchange Components (BIECs)*, because they are either specific to an organization's business or they represent a more general line of business that crosses organizational lines. Often they are business data components developed and used by multiple organizations within the same community of interest. So, instead of an *organization*, it is more appropriate and provides better context if we use the term *information sharing enterprise*.

> **[Definition: Information Sharing Enterprise]**
>
> A group of organizations with business interactions that agree to exchange information, often using multiple types of information exchanges. The member organizations have similar business definitions for objects used in an information exchange and can usually agree on their common BIEC names and definitions.

Information sharing enterprises may cross various levels of government and involve multiple business domains. They may be self-defining and can be formal (with specific governance) or informal and *ad hoc*. An information sharing enterprise is the primary entity that supports the development and management of BIECs and an associated Enterprise Information Exchange Model (EIEM) (to be discussed next). Henceforth, unless otherwise

stated, all references to an enterprise will implicitly mean information sharing enterprise.

A *Business Information Exchange Component (BIEC)* **[NIEM BIEC]** is a NIEM-conformant data component in XML Schema that meets the specific business needs of an information sharing enterprise for exchanging data about something that is a part of one or more information exchanges. This data component is tailored and intended to be used consistently across multiple IEPDs built by an enterprise. A BIEC is a NIEM-conformant data component that is:

- Reused from a NIEM release (for example, as a subset; with possibly modified cardinality), or
- Extended per the **[NIEM NDR]** from an existing NIEM data component, or
- Created per the **[NIEM NDR]** as a new data component that does not duplicate existing NIEM components within a release in use.

[Definition: Business Information Exchange Component (BIEC)]

A NIEM-conformant XML schema data component definition or declaration (for a type, element, attribute, or other XML construct) reused, subsetted, extended, and/or created from NIEM that meets a particular recurring business requirement for an information sharing enterprise.

The use of BIECs has the potential for simplifying IEPD development and increasing consistency of the business object definitions at all steps in the process, including exchange content modeling, mapping to NIEM, creating NIEM extension components, and generating XML schema documents.

An *Enterprise Information Exchange Model (EIEM)* is an MPD that incorporates BIECs that meet enterprise business needs for exchanging data using **[NIEM BIEC]**s. An EIEM is an adaptation of NIEM schema documents, tailored and constrained for and by an enterprise. An EIEM contains the following schema documents that are commonly used or expected to be used by the authoring enterprise:

- One standard NIEM schema document subset (or reference schema document set).
- Optionally, one or more NIEM extension schema documents that extend existing NIEM data components or establish new NIEM-conformant data components.
- Optionally, one or more NIEM constraint schema document sets (usually based on a schema document subset).
- Optionally, one or more XML schema documents for non-NIEM (i.e., non-conformant) standards with associated extension schema documents that contain adapter types for the data components that will be used from those non-NIEM XML schema documents (per **[NIEM NDR]**).

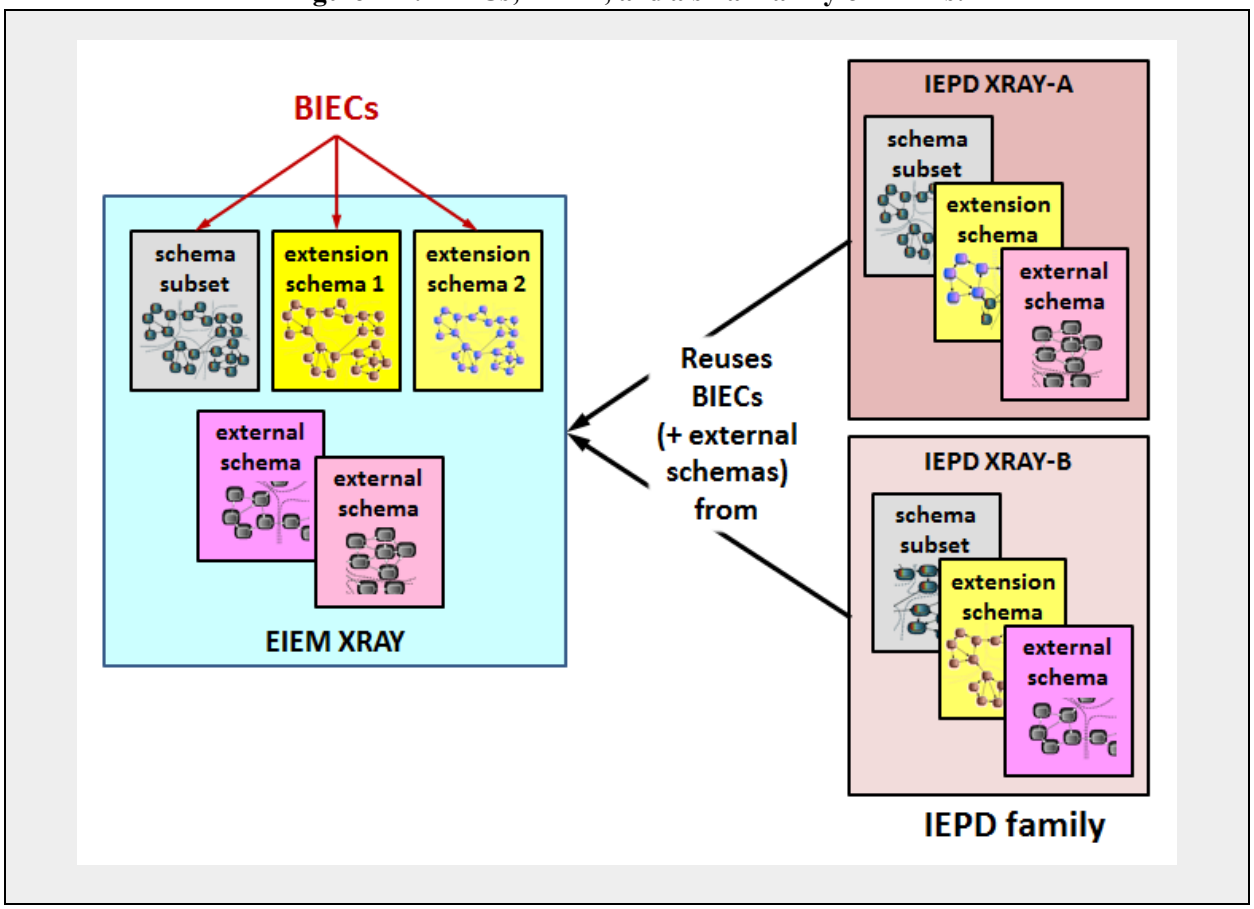
[Definition: Enterprise Information Exchange Model (EIEM)]

An MPD that contains a NIEM-conformant schema document set that defines and declares data components to be consistently reused in the IEPDs of an enterprise. An EIEM is a collection of BIECs organized into a schema document subset. As options, an EIEM can include extension, constraint, and external(non-NIEM) XML schema documents.

An information sharing enterprise that creates and maintains an EIEM authors IEPDs by reusing its EIEM content instead of (re)subsetting reference schema documents sets and (re)creating extensions. An EIEM may also contain business rules or constraint schema document sets tailored to enterprise requirements and designed to restrict variability in use of NIEM data components. This not only saves time, but it also ensures that enterprise IEPDs reuse NIEM and associated extensions consistently. (XML schema document subsets, extension schema documents, and constraint schema document sets will be defined and discussed in more detail later in this document). Figure 2-2, *BIECs, EIEM, and a small family of IEPDs.*, below, generally illustrates how BIECs, an EIEM, and an IEPD family relate (Constraint schema document sets are optional and not depicted in this

figure).

Figure 2-2: BIECs, EIEM, and a small family of IEPDs.



2.11. Similarities and Differences of MPD Classes

It will be helpful to summarize the foregoing discussions by listing the primary similarities and differences among the various types of MPDs. This will help highlight the nature of this specification as a baseline and point of leverage for all five classes of MPDs: NIEM release, core update (CU), domain update (DU), IEPD, and EIEM. Note that these lists are not all inclusive.

MPD class similarities:

- Principal artifacts are XML schema documents (XSD), the purpose for which is to define and declare reusable data components for information exchanges or to define the exchanges themselves.
- Each MPD requires a self-documenting `mpd-catalog.xml` artifact containing metadata and a listing of its key artifacts. This artifact establishes MPD name, version, class, purpose, general content, lineage, etc.
- Each MPD requires a Uniform Resource Identifier (URI) and a version number.
- Each MPD is packaged as a self-contained **[PKZIP]** archive. Self-contained simply means that an MPD has copies of (not just URLs or references to) all schema documents needed to validate instance XML documents it defines.
- Each MPD may contain optional alternate representations in addition to XML Schema (for example, generic diagram, UML diagram, XMI, database format, spreadsheet, etc.).

MPD class differences:

- IEPDs and EIEMs contain subset, extension, external, and constraint schema documents and document sets. NIEM releases, core updates, and domain updates contain reference and external schema document sets.

- An IEPD must declare at least one ·IEP conformance target· within its MPD Catalog Section 4.1, *NIEM MPD Catalog*, below. Other MPD classes do not have this requirement.
- IEPDs, EIEMs and domain updates may optionally contain sample instance XML documents and associated XSLT files to display them.
- A domain update may supersede and replace another published schema document/namespace. It may also add to or modify content in another published schema document/namespace without including the unchanged content. Core updates may only add to or supplement; and never replaces or modifies a NIEM Core.
- IEPDs, EIEMs, and NIEM releases are independently complete. A Core update can be issued as a new complete standalone reference schema document to be used with a NIEM Core.

Table 2-2, *Comparison of MPD classes*, below, summarizes many of the similarities and differences of MPD classes by indicating characteristics for each:

Table 2-2: Comparison of MPD classes

Characteristics of MPD Classes	Release	CU	DU	IEPD	EIEM
Requires a URI	X	X	X	X	X
Requires a version number	X	X	X	X	X
Must be packaged as a **[PKZIP]** archive	X	X	X	X	X
May contain alternate model representations (in addition to XSD)	X	X	X	X	X
Requires an `mpd-catalog.xml` artifact (specified by XSD)	X	X	X	X	X
Requires a formal XML change log (specified by XSD)	X	X	X		
Requires a read-me artifact				X	X
Its XML schema document set defines reusable data components	X	X	X		X
Its XML schema document set defines data exchanges (IEPs)				X	
Can contain subset, extension, external, or constraint schema documents				X	X
Contains reference and external schema documents only	X	X	X		
Must declare at least one or more ·IEP conformance targets·				X	
May contain sample instance XML documents that validate to XML schema document set			X	X	X
Required to be independently complete standalone XML schema document set	X			X	X

3. MPD XML Schema Document Artifacts

·XML schema document· artifacts are the essential content of MPDs because they normatively define and declare data components. The purpose of an MPD is determined by the ·XML schema document· or document set(s) it contains; furthermore, each · XML schema document· may have a different purpose. The **[NIEM NDR]** addresses some schema documents as ·conformance targets· including reference schema documents, extension schema documents, and schema document sets. Each conformance target may adhere to a different (though possibly overlapping) set of conformance rules. Consult the **[NIEM NDR]** for these rules. NIEM also employs a special technique that relies on ·constraint schema documents· and document sets.

The following subsections will define each type of NIEM schema document and document set, and will identify the types of MPDs that contain them.

3.1. Reference Schema Documents

This section generally applies to NIEM releases, core updates, and domain updates. Though not common, it is also valid to use a ·reference schema document· or document set within an IEPD or EIEM. The ·reference schema document· and ·reference schema document set· were defined earlier in Section 2.7, *Reference Schema Documents*, above.

A NIEM ·reference schema document· is intended to be the authoritative definition schema document for a NIEM target namespace, therefore, all NIEM releases, core updates, and domain updates are composed of a reference schema document set and associated namespaces. As a standalone artifact set, a reference schema document set is always harmonized such that all types and properties are semantically unique (i.e., multiple versions of semantically identical types or properties do not exist within the set).

As authoritative definitions, NIEM reference schema document sets satisfy more rigorous documentation requirements. The **[NIEM NDR]** requires that each type definition, and element and attribute declaration in a reference schema document contain an `xs:annotation` element that defines its semantic meaning. As will be explained later, ·extension schema documents· are also authoritative definitions, but in a local sense. They are authoritative within a given IEPD or EIEM, and therefore, must also satisfy the same rigorous documentation rules as reference schema documents.

Typically reference schema documents contain data components with the most relaxed cardinality (`0` to `unbounded`). However, this is not an absolute requirement. If necessary, cardinality in reference schema documents may be constrained to model reality. For example, in NIEM 3.0 a `nc:Location2DGeospatialCoordinateType` contains both a `nc:GeographicCoordinateLatitude` element and a `nc:GeographicCoordinateLongitude` element. Each of these elements has cardinality `minOccurs="1"` and `maxOccurs="1"`. Any other cardinality for these elements has no meaning. On the other hand, one might claim that NIEM should constrain `PersonType` to a single occurrence of the element `PersonBirthDate`. Every person has one and only one birth date. Unfortunately, also in reality, criminal persons often present multiple identities with multiple birth dates; and so the capability to represent such is an important data requirement for NIEM.

3.2. Subset Document Schemas

This section applies to IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain schema document subsets (only reference schema document sets).

3.2.1. Basic Subset Concepts

A NIEM *schema document subset* is a set of XML schema documents that constitutes a reduced set of components derived from a NIEM reference schema document or document set associated with a given numbered release or domain update. Any given XML schema document within a schema document subset is referred to as a *subset schema document* (terms reversed).

[Definition: subset schema document]

An XML schema document that meets all of the following criteria:

- It is built from a reference schema document set where one or more reference schema documents has been substituted by a its corresponding subset schema document.
- It is built from a reference schema document by applying subset operations to the XML schema statements in a reference schema document.
- It is explicitly designated as a subset schema document. This is accomplished by declaration in the relevant MPD catalog or by a tool-specific mechanism outside the subset schema document.
- It has a target namespace previously defined by a reference schema document. That is, it does not provide original definitions and declarations for schema components, but instead provides an alternate schema representation of components that are defined by a reference schema document.
- It does not alter the business semantics of components in its namespace. The reference schema document defines these business semantics.

- It is intended to express the limited vocabulary necessary for an IEPD or EIEM and to support XML Schema validation for an IEPD.

See also ·schema document subset·.

The primary purpose for a schema document subset is to reduce and constrain the scope and size of a full NIEM reference schema document set for use within an IEPD or EIEM. Thus, a schema document subset is derived from a reference schema document set (such as a NIEM release) by applying subset operations (See Section 3.2.2, *Subset Operations*, below). Also, note that the process of deriving a schema document subset from a NIEM reference schema document set is optional; it is completely valid to reuse NIEM reference schema documents as-is within IEPDs or EIEMs.

[Definition: schema document subset]

An XML schema document set built from a reference schema document set by applying subset operations to the reference schema documents in that set.

See also ·subset schema document·.

Because NIEM adopts an optional and over-inclusive data representation strategy, most elements in a NIEM reference schema have zero to unbounded cardinality. So, elements with cardinality `minOccurs="0"` are optional and may be omitted from a subset schema document if not needed for business reasons. It is also valid to constrain element cardinality within a subset schema document, as long as doing so does not break the subset relationship with the reference schema document set. For example, a reference schema document element with cardinality (`minOccurs="0"`, `maxOccurs="unbounded"`) may be constrained to (`0`,`1`) or (`1`,`1`) in a subset schema document. However, if a reference schema document element's cardinality is (`1`,`unbounded`), it may not be constrained to (`0`,`1`) since this breaks the subset relationship. The interval (`0`,`1`) is not contained within, and instead, overlaps the interval (`1`,`unbounded`).

The fundamental rule for a valid schema document subset is as follows:

[Rule 3-1] (Subset) (Constraint)

A subset schema assembled from a ·schema document subset· derived from a NIEM ·reference schema document set· is a valid subset schema, if and only if any instance XML document that validates with the schema subset also validates with the reference schema assembled from the ·reference schema document set·.

3.2.2. Subset Operations

This section applies to IEPDs and EIEMs.

NIEM subset operations are essentially reduction operations that remove or constrain portions of a reference schema document set, thereby building a profile of the set. They do not expand the scope (i.e., relax constraints) or change the semantics of reference schema document set content.

The following describe valid operations that, if applied to a schema document set will result in a corresponding ·schema document subset·:

1. Remove an XML comment.

2. Remove an `xs:annotation` and its children `xs:documentation` and `xs:appinfo`.
3. Increase the value of an `xs:element/@minOccurs` as long as it remains less than or equal to its corresponding `@maxOccurs` value).
4. Decrease the value of an `xs:element/@maxOccurs` as long as it remains greater than or equal to its corresponding `@minOccurs` value.
5. Remove an `xs:element` if its `@minOccurs="0"`.
6. Remove an `xs:complexType` or `xs:simpleType` if not supporting an `xs:element` or `xs:attribute` declaration, or another `xs:complexType` or `xs:simpleType` definition.
7. Remove an `xs:attribute` with `@use="optional"` from an `xs:complexType`.
8. Change an `xs:attribute/@use="optional"` to `@use="prohibited"`.
9. Change an `xs:attribute/@use="optional"` to `@use="required"`.
10. Remove an `xs:element` declaration if it is not supporting an element use.
11. Remove an `xs:enumeration` from an `xs:simpleType` as long as it is not the only remaining `xs:enumeration`.
12. Remove an element with representation term `AugmentationPoint` if it is not being used for element substitution.
13. Add or apply a constraining facet to an `xs:simpleType`.
14. Remove an `xs:import` and its associated schema document (if the schema document is not used within the document set).
15. Change a concrete `xs:element` declaration to `@abstract="true"`.
16. Change an `xs:element/@nillable="true"` to `@nillable="false"`.
17. Substitute an `xs:element/@substitutionGroup` member for its associated substitution group head.
18. Substitute a composition of `xs:element/@substitutionGroup` members for their associated substitution head (subject to cardinality and unique particle attribution (UPA) constraints **[W3C XML Schema Structures]**, §Schema Component Constraint: Unique Particle Attribution). The composition is an ordered sequence of the `@substitutionGroup` member elements. Each substitute element may bound its cardinality such that the total cardinality sum is within the bounds of the `@substitutionGroup` head cardinality. Order and cardinality of the replacement sequence must conform to XML Schema UPA constraints.
19. Replace a wildcard (subject to cardinality, UPA, and namespace constraints) with a composition, i.e., an ordered sequence of elements. Each element may further bound cardinality within the bounds of the wildcard. Order and cardinality of replacement sequence must conform to XML Schema UPA constraints. The namespace of each element must conform with namespace constraints specified by the wildcard (if any).

3.2.3. Subset Schema Document Namespaces

This section applies to IEPDs and EIEMs.

A ·schema document subset· is essentially a ·reference schema document set· (i.e., a numbered release) that has been modified by applying the foregoing subset operations Section 3.2.2, *Subset Operations*, above, to support business requirements represented in an IEPD or EIEM. A subset derived from a reference schema document set may differ from that reference set only in that its content has been reduced and/or constrained. For this reason, each ·subset schema document· adopts the target namespace of its corresponding ·reference schema document·.

[Rule 3-2] (Subset) (Constraint)

A ·subset schema document· in a ·schema document subset· derived from a ·reference schema document set· MUST have the same target namespace as the schema document in the ·reference schema document set· on which it is based.

3.2.4. Multiple Schema Document Subsets in a Single IEPD or EIEM

This section applies to NIEM IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain schema subsets.

Previous sections defined a single schema subset derived from a reference schema document set. In general, an IEPD or EIEM contains a single cohesive schema subset (which may be a rather large set of files) based on one numbered NIEM release or domain update.

However, this specification does not restrict the number of different subsets that may be employed within a single IEPD or EIEM. Furthermore, it does not restrict the employment of subsets from different numbered releases within a single IEPD or EIEM. However, exercising this degree of flexibility makes it critically important that developers understand the potential consequences. NIEM subsets represent a delicate compromise between flexibility and interoperability. On the one hand, a set of IEPDs based on the same subset and numbered release use identical data components, thereby enhancing interoperability. On the other hand, mixing dissimilar subsets from the same numbered release or mixing subsets derived from various numbered releases has the potential to introduce semantic duplication and subtle ambiguity. This, in turn, can have a negatively impact on interoperability.

The NIEM mandate that every schema have a unique target namespace prevents name conflicts between reference schema document sets and between two subsets derived from different reference sets. In spite of namespace distinction, mixing subsets of multiple reference schema document sets can still introduce multiple versions of semantically equivalent data components, a potentially ambiguous situation. Even employing multiple subsets together that have been derived from the same reference set has the potential to create a similar result. Above all, it is the developer's responsibility to ensure that, if mixing subsets from one or more numbered releases within a single IEPD or EIEM, these artifacts are carefully coordinated and clearly documented to ensure the various versions of semantically equivalent data components and different schemas with the same namespaces will not cause conflicts, confusion, and/or failure during validation or exchange implementation.

3.3. Extension Schema Documents

This section applies to NIEM IEPDs and EIEMs. NIEM releases, core updates, and domain updates do not contain extension schema documents.

> **[Definition: extension schema document]**
>
> A NIEM-conformant schema document that adds domain or application specific content to the base NIEM model.

The **[NIEM NDR]** defines an IEPD ·extension schema document· as a ·conformance target·. In general, an extension schema document contains components that use or are derived from the components in reference schema documents. It is intended to express the additional vocabulary required for an IEPD, above and beyond the vocabulary available from reference schema documents.

An IEPD or EIEM developer who determines that NIEM is missing elements required for a given information exchange has several options to account for such requirement shortfalls. Using rules and techniques defined in the **[NIEM NDR]**:

- Extend an existing NIEM data component (if possible).
- Augment an existing NIEM data type (through NIEM Type Augmentation).
- Build a new NIEM-conformant data component.
- Employ NIEM adapter types for components from an external standard that does not conform to NIEM.

A NIEM extension schema document may contain data components built from any of the options above.

Employment of extension schema documents in an IEPD is entirely optional.

Multiple extension schema documents are allowed in a single IEPD. Developers will likely want to reuse many of their extension schema documents in other IEPDs. Therefore, the best practice for extension is to group all data components designed to reuse into one extension schema document or document set, and group IEPD-specific data components into another. Then the reusable extension components can be more easily redeployed in other IEPDs as needed. Also, recall that Section 2.10.5, *Enterprise Information Exchange Model (EIEM)*, above, discusses EIEM employment for larger scale reuse of NIEM data components in multiple IEPDs.

Extension schema documents generally contain new data component declarations that may (though not necessarily) be derived from or reference existing NIEM data components. This being the case, reference schema documents do not exist for new data components found within extension schema documents. Therefore, extension schema documents must satisfy the more rigorous documentation requirements of reference schema documents. Per the **[NIEM NDR]**, the definition or declaration of each new data component in an extension schema document must include an `xs:annotation` element that provides its semantics and NIEM-specific relationships.

3.4. External Schema Documents

In all MPD classes NIEM allows the use of *external schema documents* that do not conform to NIEM. Data components declared and defined in external schema documents require NIEM *adapter types* to identify the fact they do not conform to NIEM.

> **[Definition: external schema document]**
>
> Any XML schema document that is not a NIEM-supporting schema and that is not NIEM-conformant.

Refer to the **[NIEM NDR]** for details about external schemas, adapter types, and the rules defining their use.

3.5. Constraint Schema Documents and Document Sets

This section only applies to NIEM IEPDs and EIEMs that may use constraint schema documents or document sets. NIEM releases, core updates, and domain updates do not contain constraint schema documents. Also, note that use of constraint schemas in NIEM is perfectly valid. However, the preferred technique is to employ formal business rules.

A *constraint schema document* is an optional IEPD or EIEM artifact that is used to express business rules for a class of instance XML documents, and is not assumed to be a definition for the semantics of the components it contains and describes. Instead, a constraint schema document uses the XML Schema Definition Language to add constraints to components defined or declared by other schema documents, usually a schema document subset (but they can be applied to extension schema documents as well).

> **[Definition: constraint schema document]**
>
> An ·XML schema document· that imposes additional constraints on NIEM-conformant instance XML documents; constraint schema documents are not required to conform to NIEM.
>
> See also ·constraint schema document set·.

[Definition: constraint schema document set]

A set of related constraint schema documents that work together, such as a constraint schema document set built by adding constraints to a schema document subset.

See also ·constraint schema document·.

A constraint schema document or document set validates additional constraints imposed on an instance XML document only after it is known to be NIEM-conformant (i.e., has been validated with a reference schema document set, or schema document subset, and applicable extension schema documents).

To use a ·constraint schema document set· to tighten constraints on an IEP, a two-pass validation technique is employed. In the first pass, an IEP is validated against the schema document subset and extension schema documents. This pass ensures that IEP semantics and structure conform to the NIEM model and NDR. In the second pass, an IEP is checked against a constraint schema document set, which may contain constrained versions of the ·subset schema documents· and ·extension schema documents·. This pass ensures that the IEP also satisfies the additional constraints (i.e., business rules that the first pass was unable to validate). A constraint schema document need not validate constraints that are applied by other schema documents.

Constraint schema documents are generally useful when it is necessary to impose restrictions that are more complex than cardinality. If only cardinality restrictions are needed, then it is easier and more efficient to set these directly in the subset schema documents and avoid the use of constraint schema documents. Otherwise, constraint schema documents may be necessary. Note however, that any cardinality restrictions placed on NIEM release components within schema document subsets must not violate the rules established in Section 3.2.1, *Basic Subset Concepts*, above, which define the relationship of the reference schema document to a subset schema document derived from it.

Use of constraint schemas is one option for applying additional business rules to or tightening constraints on NIEM IEPs beyond what NIEM itself provides. This particular technique uses the XML Schema Definition Language **[W3C XML Schema Datatypes]**, **[W3C XML Schema Structures]**. NIEM also allows other methods that do not use XML Schema, such as **[ISO Schematron]** or other methods. However, at this time there are no normative rules for how these techniques should be employed in NIEM IEPDs or EIEMs. Therefore, if other techniques are used, it is a developer responsibility to incorporate appropriate artifacts and clear documentation.

Constraint schema documents are generally designed and employed in sets, similar to reference schema document set or schema document subsets. A common practice for creating an IEPD or EIEM constraint schema document set is to start with a valid NIEM schema document subset and modify it to further restrict the class of instance XML documents (IEPs) that will validate with this constraint schema set. However, an extension schema document can also be used to derive a constraint schema document. The namespace of a constraint schema document is established the same way the namespace of a subset schema document is established, by reusing the target namespace of the schema document from which it is derived.

[Rule 3-3] (Constraint) (Constraint)

A ·constraint schema document· MUST have a target namespace that has been previously assigned to a ·reference· or ·extension schema document·, or is a ·constraint schema document· intended to support another ·constraint schema document· that has such a target namespace.

There is no restriction on the number of constraint schema document sets that an IEPD or EIEM can employ. As in other advanced situations, developers must clearly document their intentions for and use of multiple constraint

schema document sets.

In general, ·constraint schema documents· have far fewer requirements than other classes of NIEM schema documents. Since they work in tandem with NIEM normative schema documents, constraint schema documents are allowed to use the XML Schema Definition language in any way necessary to express business rules. This means that to constrain instance XML documents, constraint schema documents can employ XML Schema constructs that are not allowed in other classes of NIEM schema documents.

BIECs in particular may have additional business rules in constraint schema documents. A normative NIEM BIEC Specification (not available at the time of the publication of this MPD Specification), will supplement or obviate constraint schema documents with consistent and formal techniques for representing business rules within NIEM components. However, as already mentioned, the MPD Specification does not prohibit or restrict the application of formal business rule techniques (such as **[ISO Schematron]** or any other form of business rules) to MPDs now.

3.6. Classes of MPDs vs. Classes of Schema Documents

The chart in Table 3-1, *Schema document classes vs. MPD classes*, below, summarizes the types of schema documents that: (1) can be contained in an instance of each MPD class, and (2) the (*minimum, maximum*) cardinalities of those schema documents. In some cases, certain types of schema documents are never contained in particular class of MPD. These are labeled *N/A*, i.e. *not applicable*.

Notice that only NIEM releases, core updates, and domain updates contain reference schema document sets, while only IEPDs and EIEMs contain the user-developed schema document sets. Plus (+) indicates that a NIEM-conformant IEPD or EIEM should contain and use at least one schema document that is either a NIEM ·reference schema document· or a NIEM ·subset schema document· from such.

Table 3-1: Schema document classes vs. MPD classes

Schema Document Classes	Release	CU	DU	IEPD	EIEM
Reference	(1, unbounded)	(1, unbounded)	(1, unbounded)	(0+, unbounded)	(0+, unbounded)
Subset	N/A	N/A	N/A	(0+, unbounded)	(0+, unbounded)
Constraint	N/A	N/A	N/A	(0, unbounded)	(0, unbounded)
Extension	N/A	N/A	N/A	(0, unbounded)	(0, unbounded)
External	(0, unbounded)	(0, unbounded)	(0, unbounded)	(0, unbounded)	(0, unbounded)

4. MPD Documentation Artifacts

XML schema documents (and the schemas that result from them) are the essence of a NIEM MPD. All other artifacts are considered documentation.

A variety of documentation files may be incorporated into a NIEM MPD. However, in addition to XML schema documents, there are only two mandatory documentation artifacts required by every MPD: the *mpd-catalog* and the *change log*. An MPD catalog (`mpd-catalog.xml`) contains basic metadata, relationship and lineage data, and validation information. The change log provides a history of modifications.

A ·read-me artifact· (formerly known as a *master document*) is mandatory for IEPDs and EIEMs. These MPD classes may be built by different developers, and may be registered into a repository for reuse by many other users, developers, and implementers; therefore, a minimal form of documentation is absolutely necessary. An

IEPD or EIEM read-me file is the primary source and starting point for human readable documentation, and should reference (and describe) any other separate documentation artifacts. This requirement ensures that baseline documentation is consistently rooted in a clearly visible artifact within each IEPD and EIEM.

The following subsections will address these artifacts and the concepts, metadata, and content each supports.

4.1. NIEM MPD Catalog

[Definition: MPD catalog document]

An XML document that is Schema valid to `mpd-catalog-3.0.xsd` in Appendix A, *MPD Catalog XML Schema Document*, below. The *MPD catalog document* contains metadata that describes:

- Unique identification
- Basic characteristics and properties
- Key artifacts and directory structure
- Relationships to other MPDs and their artifacts
- ·IEP conformance targets·

All MPDs require an `mpd-catalog.xml` artifact. So, the ·MPD catalog document· is tailored to accommodate all MPD classes. However, each MPD class has different catalog requirements. The catalog metadata are formally defined and declared in an XML schema Appendix A, *MPD Catalog XML Schema Document*, below.

This metadata is designed to be the minimal needed to facilitate human understanding, tool support, and machine processing. The metadata can support a number of MPD uses and functions including (but not limited to):

- Identification of key artifacts
- Generation of a hyperlinked content display using XSLT
- Browsing and understanding of artifacts and their content
- Automatic registration into a registry/repository
- Search, discovery, retrieval of MPDs (through metadata and relationships)
- Reuse of MPDs and their artifacts
- Reuse of BIECs and associated EIEMs
- Tracing and analysis of MPD lineage
- General conformance and validation of the MPD itself
- Definition, identification, and validation of IEP conformance targets

[Rule 4-1] (MPD-Cat) (Constraint)

An ·MPD catalog document· MUST have an artifact named `mpd-catalog.xml` in its ·MPD root directory·.

[Rule 4-2] (MPD-Cat) (Constraint)

An ·MPD catalog document· MUST be valid to `mpd-catalog-3.0.xsd` Appendix A, *MPD Catalog XML Schema Document*, below.

This rule only requires validation with `mpd-catalog-3.0.xsd` and does not require that this schema document be contained in an MPD. Tools that validate `mpd-catalog.xml` will all access the same schema document to perform such.

> **[Rule 4-3] (MPD-Cat) (Constraint)**
>
> An ·MPD catalog document· MUST be valid to the NIEM MPD catalog Schematron rules in Appendix C, *Schematron Rules for MPDs*, below.

Note that Appendix A, *MPD Catalog XML Schema Document*, below, is a fairly relaxed XML schema definition with very few mandatory properties. The reason for this is to support tools that must identify and distiguish a ·well-formed· from a ·NIEM IEPD· during incremental stages of development. This is explained in more detail in Section 5.1, *Well Formed MPD (WF-MPD)*, below.

Because the baseline XML Schema definition for an MPD catalog is more relaxed than is required for a ·NIEM IEPD·, special Schematron rules also apply to an MPD catalog to ensure its validity or completeness.

4.1.1. NIEM Core Schema Document Subset that Supports MPD Catalog

Appendix A, *MPD Catalog XML Schema Document*, below, is a NIEM-conformant schema document. It reuses a schema document subset of NIEM 3.0 (primarily from NIEM Core). **[NIEM MPD Toolkit]** contains the NIEM 3.0 schema document subset used by `mpd-catalog-3.0.xsd`.

As is the case with `mpd-catalog-3.0.xsd`, MPDs are not required to contain this schema document subset since it is always available here: `http://reference.niem.gov/niem/resource/mpd/3.0/`

4.1.2. MPD Catalog as a Table of Contents

One function of the MPD catalog is to serve as a table of contents that identifies, locates, and classifies key artifacts and artifact sets. For that purpose Appendix A, *MPD Catalog XML Schema Document*, below, provides a number of classifier elements for most common artifacts and artifact sets in MPDs. For other less common or generic artifacts two general classifiers exist: `c:Documentation` and `c:ApplicationInfo`. These elements loosely correspond to the meaning of the XML Schema `xs:annotation` child elements, `xs:documentation` and `xs:appinfo`. General visual, audio, and textual explanatory documentation should be classified as `c:Documentation`, while tool-specific artifacts (such as imports, exports, executables, etc.) should be classified as `c:ApplicationInfo`.

The classifier elements are designed to identify, categorize, and describe any artifacts and artifact sets (including its path name, dependencies, and lineage). Employing XSLT, `mpd-catalog.xml` can be transformed into an `index.html` artifact that displays a hyperlinked MPD table of contents and metadata summary in a browser.

In general, only an IEPD or EIEM would contain `c:Documentation` and `c:ApplicationInfo` artifacts. So, for an IEPD and EIEM, a best practice is to use the `read-me` artifact (i.e., the ·read-me artifact· required in the ·MPD root directory·) to reference `c:Documentation` and `c:ApplicationInfo` artifacts whether or not they have been classified in the MPD catalog.

Release, core update, and domain update catalogs are required to record all artifacts. However, IEPD and EIEM MPD catalogs are not. The IEPD or EIEM author decides which artifacts (both files and sets) are important enough to explicitly include in the MPD catalog. The author may choose to classify all, some, or none in the catalog.

The MPD catalog provides a supplement or an alternative to organizing MPD artifacts and sets with a standard

file directory. An author can use it to identify, classify, and describe particular artifacts or sets, instead of having to do so with only file names and directory structure. An author can also employ the guidance in Appendix F, *Guidance for IEPD Directories (non-normative)*, below.

4.1.3. Extending an MPD Catalog

An MPD Catalog may be extended to accommodate new or additional metadata, artifact classifiers, or validity constraints that are not already defined in Appendix A, *MPD Catalog XML Schema Document*, below.

To extend the MPD catalog, an MPD author must provide both an XML catalog extension document (XML) and one or more MPD extension schema documents (XSD). The XML catalog extension identifies that one or more MPD catalog extensions are present, and resolves their namespaces to local URIs. The MPD catalog extension is a schema that defines and declares the new data components for metadata, classifiers, and/or constraints. Both general **[NIEM Conformance]** and specific **[NIEM NDR]** conformance rules apply to these components. The XML catalog extension document must reside in the ·MPD root directory·. The MPD extension schema documents may bear any file name and reside anywhere in the MPD. This is because the XML catalog is expected to ·resolve· all local URIs. MPD processing tools are expected to look for and recognize the XML catalog (that identifies MPD catalog extensions exist) by its file name.

The following rules specify the requirements for an MPD catalog extension XML catalog document:

[Rule 4-4] (MPD-Cat) (Constraint)

An MPD catalog extension XML catalog document MUST reside in the same relative directory as the `mpd-catalog.xml` artifact (normally in the ·MPD root directory·)

[Rule 4-5] (MPD-Cat) (Constraint)

An MPD catalog extension XML catalog document MUST bear the file name (and type) `mpd-catalog-extension-xml-catalog.xml`.

[Rule 4-6] (MPD-Cat) (Constraint)

An MPD catalog extension XML catalog document MUST ·resolve· all MPD catalog schema extension document namespaces to the correct corresponding local URIs in the MPD.

So, when a processor identifies a file named `mpd-catalog-extension-xml-catalog.xml` in the ·MPD root directory·, it can assume that it contains references to one or more MPD catalog extension schema documents that adhere to the following rules:

[Rule 4-7] (MPD-Cat) (Constraint)

An MPD catalog extension schema document MUST conform to the **[NIEM NDR]** extension schema conformance target rules.

[Rule 4-8] (MPD-Cat) (Constraint)

Within an MPD, the schema set formed by `mpd-catalog-3.0.xsd` and all MPD catalog extension schema documents MUST conform to the **[NIEM NDR]** schema set conformance target rules.

Whether extending an MPD catalog with new metadata elements, artifact classifier elements, or validity constraint elements, Appendix A, *MPD Catalog XML Schema Document*, below, provides an abstract element as a substitution group head in each case. The user simply derives a new type (through extension or restriction), or reuses an existing type, then declares a new element (of that type), and identifies it with the appropriate substitution group. Whenever possible, the user should reuse types, elements, and attributes that are already defined/declared within the Appendix A, *MPD Catalog XML Schema Document*, below.

If an MPD catalog schema document extension uses NIEM data components that are not already contained in the NIEM Core subset provided with **[NIEM MPD Toolkit]**, then the additional components must be additive. In other words:

[Rule 4-9] (MPD-Cat) (Constraint)

Any subset schema documents provided to support an MPD schema document extension MUST be a superset of the subset schema documents provided with this specification to support the MPD catalog schema document.

4.2. Metadata Concepts

The MPD catalog also contains both required and optional metadata for the MPD and its artifacts and artifact sets. The following subsections specify the syntax, formats, and semantics for that metadata.

4.2.1. Version Numbering Scheme

Published MPDs may be periodically revised and updated; therefore, versioning is required to clearly indicate changes have occurred. In order to maintain some consistency while allowing reasonable flexibility to authors, this specification establishes a simple version numbering scheme that is consistent with most common practices. This is the same version numbering scheme that is used for NIEM releases.

[Rule 4-10] (WF-MPD) (Constraint)

An MPD MUST be assigned a version number that adheres to the regular expression:

```
version ::= digit+ ('.' digit+)* (status digit+)?
Where:
        digit   ::= [0-9]
        status  ::= 'alpha' | 'beta' | 'rc' | 'rev'
```

The meaning of the `status` values are as follows:

- `alpha` indicates early development; changing significantly.
- `beta` indicates late development; but changing or incomplete.
- `rc` indicates release candidate; complete but not approved as operational.
- `rev` indicates very minor revision that does not impact schema validation.

The regular expression notation used above is from **[W3-XML]** `#sec-notation`.

Note that the absence of a `status` string in the version number indicates that the version has been baselined and published.

The regular expression in Rule 4-10, above, is enforced by the MPD catalog schema document Appendix A, *MPD Catalog XML Schema Document*, below. The following examples are valid MPD version numbers:

- `1`
- `1.2`
- `1.3.1.0`
- `1.2alpha13`
- `199.88.15rev6`

There are two implications in Rule 4-10, above. The first is that in some cases this version scheme implies and confirms a chronology of releases. For example, a given product labeled version 2.3 must have been released before the same product labeled 2.3.1. Therefore, version 2.3.1 is more current than version 2.3.

However, this is a multi-series version scheme, and chronological relationships exist only within a given series. So, for example, nothing can be said about a chronological relationship between versions 2.2.4 and 2.3. This is because version 2.2.4 is in a different series (i.e., 2.2) and could actually have been released after 2.3. Figure 4-1, *Example versioning system*, below, illustrates a system of versions that uses the numbering scheme of Rule 4-10, above.

Figure 4-1: Example versioning system

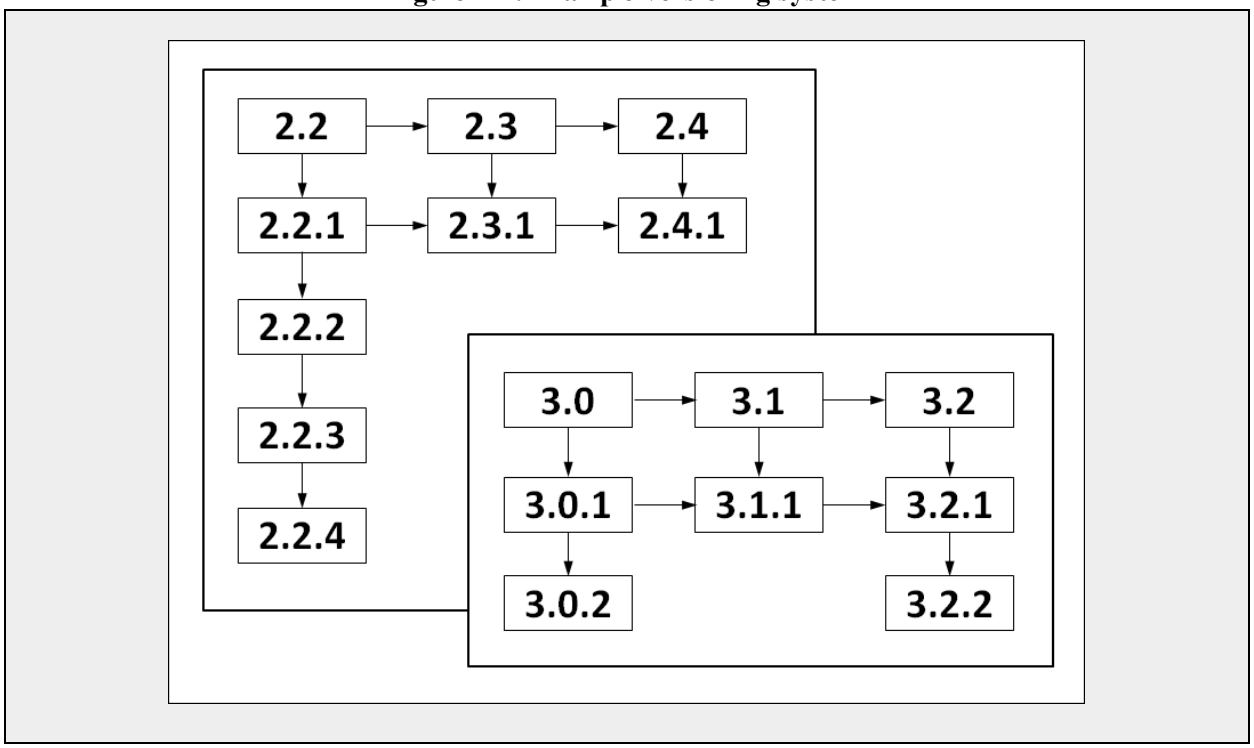


Figure 4-1, *Example versioning system*, above, illustrates eight different version series. Within this illustration these are the only sequences that have chronological relationships that can be identified through version numbers.

- Series 2 is {2.2, 2.3, 2.4}
- Series 3 is {3.0, 3.1, 3.2}
- Series 2.2 is {2.2(.0), 2.2.1, 2.2.2, 2.2.3, 2.2.4}
- Series 2.3 is {2.3(.0), 2.3.1}
- Series 2.4 is {2.4(.0), 2.4.1}
- Series 3.0 is {3.0(.0), 3.0.1, 3.0.2}

- Series 3.1 is {3.1(.0), 3.1.1}
- Series 3.2 is {3.2(.0), 3.2.1, 3.2.2}

The second implication of Rule 4-10, above, is that pre-releases are easily identified by the strings `alpha`, `beta`, and `rc`. These strings are simple visible indicators of MPD status or stage of development.

This specification places no further restrictions or meaning (implied or otherwise) on a version number. Authors have the option to use integers between dots to indicate degree of compatibility or other relationships between versions as needed. For example, for a given MPD, the author may declare that if an instance validates to version 4.2.3, then it will also validate to version 4.2. Such a claim is acceptable. However, this specification does not imply any such relationships. Any meaning assigned to version sequence by an authoritative source should be unambiguously documented within the MPD.

MPD version numbers within a version series do NOT imply compatibility between versions. Compatibility between or among MPD versions MUST be explicitly stated in documentation.

Note that an author who updates an existing MPD to a new version may choose the version number based on its previous version number or not, as long as it follows the version number syntax.

Version number syntax applies to MPDs only; there is no requirement to apply this syntax to artifact versioning.

4.2.2. URI Scheme for MPDs

To facilitate MPD sharing and reuse the assignment of a URI (Uniform Resource Identifier) to each MPD is essential. This is enforced by the MPD catalog schema document Appendix A, *MPD Catalog XML Schema Document*, below. However, it is also important to ensure that an MPD URI is absolute.

> **[Rule 4-11] (WF-MPD) (Constraint)**
>
> In an MPD catalog document, the value of a `c:mpdURI` attribute of type `xs:anyURI` MUST match the production <absolute-URI> as defined by **[RFC 3986 URI]**, §4.3, Absolute URI.

Rule 4-11, above, implies that a URI assigned to an MPD must be valid. Furthermore, the entity (person or organization) assigning the MPD URI either (1) *is* the registrant of the domain name or namespace identifier, or (2) *has* authority from the registrant to assign this URI.

Examples of valid MPD URIs:

- `http://example.gov/niem-iepd/prescription-monitoring-info-exchange/3.0/`
- `http://example.gov/niem-iepd/pmix/3.0/`
- `http://release.niem.gov/niem/niem-core/3.0/`
- `http://niem.gov/niem/domains/cyfs/2.1/1`

This specification does not mandate that basic MPD catalog metadata be designed into an MPD URI. However, including such can obviously provide for convenient visual recognition of an MPD. That said, an author should ensure any metadata embedded in the URI accurately reflect the MPD catalog metadata (in particular, the values of `c:mpdURI`, `c:mpdName`, `c:mpdVersionID`, and `c:mpdClass` defined in Appendix A, *MPD Catalog XML Schema Document*, below).

4.2.3. URI Scheme for MPD Artifacts

Artifacts in other MPDs can be referenced from within an MPD to identify equivalence (reuse) and one aspect of

lineage. To support this concept, the following MPD URI rules are necessary:

[Rule 4-12] (WF-MPD) (Interpretation)

A valid MPD URI MUST support the inclusion of a *fragment identifier* (as a suffix) **[RFC 3986 URI]**.

[Rule 4-13] (WF-MPD) (Constraint)

A valid MPD URI MUST NOT contain a *fragment identifier* **[RFC 3986 URI]**.

Rule 4-12, above, is required to ensure that any MPD can always uniquely identify and refer to each artifact within another MPD (not just an MPD). This specification follows **[RFC 3986 URI]** and employs a fragment identifier to construct a URI for an artifact within an MPD. By the following rule, each file artifact or artifact set is uniquely identified by its path name relative to the ·MPD root directory·.

[Rule 4-14] (WF-MPD) (Constraint)

Within an MPD a URI reference to an artifact in another external MPD is the concatenation of

- The URI of the MPD that contains the artifact.
- The pound-sign character ("#") (also known as the hashtag character).
- An identifier that is the artifact's locally unique path name relative to the ·MPD root directory·.

An artifact always has a locally unique path name that terminates in (and includes) its file name; an artifact set always has a locally unique path name (that terminates in and includes a slash character — `"/"`).

You should now understand the rationale for Rule 4-13, above. If an MPD URI is constructed with a fragment identifier, then that URI cannot be employed as an MPD artifact URI because **[RFC 3986 URI]** allows only a single fragment identifier.

The following are examples of valid MPD artifact URIs:

- `http://example.gov/niem-iepd/pmix/3.0/#subset/niem-core.xsd` (a file artifact)
- `http://example.gov/niem-iepd/pmix/3.0beta2/#extension/ext-1.1.xsd` (a file artifact)
- `http://example.gov/niem-iepd/pmix/3.0/#application-info/` (a set artifact)
- `http://example.gov/niem-iepd/pmix/3.0/#iep-sample/query/` (a set artifact)

Note that **[RFC 2141 URN Syntax]** appears to preclude the use of a URN as an MPD URI. URN syntax violates Rule 4-12, above, because it does not support fragment identifiers.

Here is one simple scenario for using an artifact URI within the MPD catalog. Consider two different IEPDs with the following URIs:

1. `http://example.gov/niem-iepd/pmix/3.0/`
2. `http://www.abc.org/niem-iepd/order/2.1.2rev3/`

The author of IEPD (2) has decided to reuse (as-is) the `extension/req1.xsd` artifact in IEPD (1). He/she can optionally create an MPD catalog `c:ExtensionSchemaDocument` entry for this artifact (assuming it is an extension schema document), and add the attribute:

```
c:externalURI="http://example.org/niem-iepd/pmix/3.0/#extension/req1.xsd"
```

Additional `c:externalURI` attributes can be optionally added to this entry if the author knows of other uses of this same artifact in other MPDs and wishes to acknowledge them.

Note that a URI does not have the same meaning as namespace. NIEM namespaces cannot be used as artifact URIs. Recall that the namespaces used in a schema document subset derived from a NIEM release are identical to the namespaces of the release itself. Furthermore, an IEPD or an EIEM may contain multiple subsets. NIEM namespaces are not necessarily unique to an artifact within an MPD.

Later, Section 4.5, *XML Catalogs*, below, will describe the use of **[XML Catalogs]** to correlate and ·resolve· namespaces to their corresponding local URIs.

4.2.4. MPD and MPD Artifact Lineage

An important business requirement is transparency of MPD lineage. Data lineage is also referred to as *data provenance*, how data are derived or where it came from. There are two basic views of data provenance: (1) as data annotations; and (2) as a graph of data relationships **[Principles of Data Integration]**, Chapter 14 "Data Provenance".

The MPD Specification adapts the latter view of data provenance to enable a simple framework for recording MPD lineage within an MPD catalog. The URI scheme for MPDs and their artifacts and sets enables a graph of relationships. An MPD may internally identify and record relationships to other MPDs, including families, versions, adaptations, specializations, generalizations, etc.

The MPD catalog provides a `c:Relationship` element with two attributes (`c:resourceURI` and `c:relationshipCode`) and an optional element (`nc:descriptionText`) to identify ancestry and other relationships to other MPDs. There are many ways that one MPD may relate to another. This makes it extremely difficult to specify a fixed set of values that can objectively define an exact relationship between a pair of MPDs. Therefore, the optional `nc:descriptionText` element is provided to further explain the nature of any of the eight `c:relationshipCode` values available. The set is: {`version_of`, `specializes`, `generalizes`, `deprecates`, `supersedes`, `adapts`, `conforms_to`, `updates`}. In some cases, the value of `c:relationshipCode` may be generic enough to require a more detailed explanation in `nc:descriptionText` (for example, if its value is `adapts`).

The MPD catalog also enables an author to record more fine-grained ancestry between MPDs. An MPD catalog can explicitly declare that it reuses an artifact or artifact set from one or more other MPDs. By default each artifact and artifact set identified in an MPD catalog has a globally unique URI (using a fragment reference) that can refer to it Section 4.2.3, *URI Scheme for MPD Artifacts*, above. An MPD author signifies reuse by entering the URI for the artifact or artifact set in the optional `c:externalURI` attribute within the appropriate `c:FileType` or `c:FileSetType` elements.

Some MPDs are designed for more extensive reuse than others. For example, families of IEPDs are expected to reuse a given EIEM. In such cases, the MPD catalogs for these IEPDs and the corresponding EIEM may overlap in or duplicate a large number of metadata and references. This is expected. The MPD catalog can contain many references to and semantics for artifacts, artifact sets, and MPDs. Correct and consistent use of these references and semantics will create networks of related MPDs so that tools can locate, parse, and process them as needed and when available in shared repositories.

4.3. Change Log

4.3.1. Change Log for Releases and Core/Domain Updates

Although the version identifier is useful for a fast visual indication of the state of an MPD, it only provides a

general indication that the MPD has changed. There is no indication of the volume, complexity, or impact of changes applied since a previous version. A *change log* provides a more specific accounting of changes from one version to another.

> **[Definition: change log]**
>
> A formal (for releases, core updates, domain updates) or informal (for IEPDs and EIEMs) artifact that accounts for changes applied to an MPD since its previous version (or versions).

Once published, NIEM releases always exist. This ensures that IEPDs and EIEMs built from a given release will always be usable, and may be updated to a new NIEM release only when convenient or absolutely necessary to take advantage of new or modified data components. Though not encouraged, nothing prohibits a developer from building an IEPD based on a NIEM release that is older than the most current version. There may be potential disadvantages related to interoperability levels achievable with others developing to the latest release. Nonetheless, an older version might meet the business needs of a particular organization quite well.

In spite of this built-in stability, the NIEM architecture is designed to evolve as requirements change. New versions of reference schema document sets such as NIEM releases, core updates, and domain updates can have significant impacts on future IEPDs and EIEMs. Developers must understand in detail how changes will affect their IEPD and EIEM products and the tools used to build them. To work effectively, tools for domain content development, impact analysis, migration between releases, etc. must be able to digest formal change logs. A formal change log is also essential to efficiently process and integrate new and changed content into NIEM for new releases, and to simultaneously maintain multiple versions of NIEM for users. All of the foregoing reasons dictate that NIEM require a normative change log for reference schema document sets.

> **[Rule 4-15] (WF-MPD) (Constraint)**
>
> An MPD with a `c:mpdClassCode` attribute value from the set `{rel, cu, du}` MUST contain a `changelog.xml` artifact in its ·MPD root directory·.

> **[Rule 4-16] (M-Chg-Log) (Constraint)**
>
> A `changelog.xml` artifact for an MPD with a `c:mpdClassCode` attribute value from the set `{rel, cu, du}` MUST validate with the NIEM change log schemas `mpd-changelog-1.0.xsd` and `niem-model-1.0.xsd`.

The most current versions of the resources referenced in Rule 4-16, above, are available here:

```
http://reference.niem.gov/niem/resource/mpd/changelog/

http://reference.niem.gov/niem/resource/model/
```

Since the schemas are the authority for a release or update and because almost all tool support depends on the schemas, the change log is only designed to audit transactional changes to the reference schema documents. There is no provision for logging changes to support documentation or other non-schema artifacts. Non-schema changes are generally handled non-normatively in the form of release notes.

4.3.2. Change Log for IEPDs and EIEMs

IEPD and EIEM change log requirements are less strict and are not required to conform to the naming and XML schema specifications in Rule 4-15, above. However, a change log is still required.

[Rule 4-17] (WF-MPD) (Constraint)

An MPD with a `c:mpdClassCode` attribute value from the set `{iepd, eiem}` MUST contain a change log artifact in its ·MPD root directory·

The format of an IEPD or EIEM change log is left to the discretion of the author. The use of `mpd-changelog-1.0.xsd` for IEPD and EIEM schemas is not required. Relaxing the change log format encourages and facilitates easier and more rapid development. IEPDs and EIEMs are developed by a variety of NIEM domains, organizations, and users; and they are intended to specify implementable exchanges. As a result, IEPDs and EIEMs may contain both documentation artifacts and machine readable application artifacts in a large variety of formats. A consistent standard change log would be very difficult to specify.

4.4. Read-Me Artifact

This section is applicable to IEPDs and EIEMs.

[Definition: read-me artifact]

An informal documentation artifact contained in an IEPD or EIEM that serves as the initial general source of human readable descriptive or instructional information. A *read-me* artifact or file (formerly known as a *master document*) may index or reference other more specific documentation or other explanatory materials within the MPD.

A `read-me` artifact is only required for IEPDs and EIEMs since these MPDs are allowed the greatest design flexibility, can be developed and implemented different ways, and are not centrally managed. On the other hand, releases and domain updates have restrictive rules, standard documentation for using them, and central management.

[Rule 4-18] (WF-MPD) (Constraint)

An MPD with a `c:mpdClassCode` attribute value from the set `{iepd, eiem}` MUST contain a `read-me.*` file artifact in its ·MPD root directory·.

The read-me artifact may replicate some of the metadata in the MPD catalog. However, the MPD catalog is intentionally designed to be efficient, easy to parse, and minimal. It is intended for search, discovery, registration, and Web page generation, and not to support various types of detailed technical prose often required for human understanding.

The primary purposes of the ·read-me artifact· include:

- To help facilitate understanding and reuse of IEPDs and EIEMs.
- To ensure that fundamental and detailed business-level information about an IEPD or EIEM are documented for human understanding.
- To ensure the IEPD or EIEM author has considered and conveys such fundamental information.
- To provide an initial source within an IEPD or EIEM for human consumable documentation and/or

references to other business or technical documentation needed for understanding.

The read-me artifact is not intended to be the only source of written documentation for an MPD (though it can be). It is expected to be the initial resource that references and coordinates all others whether physically present in the MPD or linked by reference. Many organizations have their own customized formats and operating procedures for documenting their work and products. This specification does not attempt to standardize read-me format or layout. Only the file name and relative path within the MPD archive are strictly specified. The following section will generally describe minimal content that should be in the read-me artifact. This guidance is non-normative, so adherence is a subjective judgment by the author.

4.4.1. Read-me Content

This section is non-normative and applies to IEPDs and EIEMs.

This section is neither a cookbook nor a normative specification for a read-me artifact. It simply suggests typical topics that a read-me artifact should or might address, and provides some non-normative guidance.

The read-me file should help another user or developer to understand the content and use of an IEPD or EIEM, as well as determine potential for reuse or adaptation. It should describe what implementers need to understand and what the author considers is important to understanding an IEPD or EIEM. There is no limit or constraint on its content.

At a minimum, the read-me file should contain several fundamental elements of information about the MPD:

- Purpose of this MPD.
- Scope of its deployment, usage, and information content.
- Business value and rationale for developing it.
- Type of information it is intended to exchange (in business terms).
- Identification of senders and receivers (or the types of senders and receivers).
- Typical interactions between senders, receivers, and systems.
- References to other documentation within the MPD, and links to external documents that may be needed to understand and implement it.

Many document formats (e.g., HTML, PDF) can display hot links to local files within the MPD archive as well as URLs to files on the Internet. Employing such a format is highly recommended but not mandatory.

[Rule 4-19] (WF-MPD) (Interpretation)

A NIEM IEPD or EIEM read-me artifact SHOULD (at a minimum) describe the MPD purpose, scope, business value, exchange information, typical senders/receivers, interactions, and references to other documentation.

MPD documentation types and formats will vary with the methodologies and tools used to develop them. Most of this documentation will likely be typical of that generated for data-oriented software projects. Some documentation may only require sections in the ·read-me artifact·. Other documentation may be more suitable as separate artifacts that are referenced and explained by a section in the read-me artifact (such as diagrams, large tables, data dictionaries, test results/reports, etc.). The following are some common examples of sections in or separate artifacts associated with the read-me artifact:

- Executive summary (especially for a lengthy read-me artifact>
- Use cases
- Business processes

- Business requirements
- Business rules
- Metadata security considerations
- Domain model design specifications and documentation and/or diagrams
- Data dictionary
- Testing and conformance
- Development tools and methodologies used
- Implementation guidance (particularly important for a complex IEPD with multiple subsets or IEP root elements)
- Security considerations
- Privacy considerations (e.g., Personal Identifiable Information)
- Types of implementations

- If an IEPD employs multiple subsets:

    - When, where, and how these are used
    - How these are coordinated in the implementation
    - Caveats regarding duplicate data components (which can occur with multiple subsets)

- If an IEPD employs multiple IEP conformance targets:

    - Purpose of each and when it should be used
    - How these are coordinated during the runtime preparation and transmission of IEPs

4.5. XML Catalogs

This section is applicable to all MPDs. However, it is of particular importance to IEPDs and IEP validation (to be covered in more detail in Section 4.6, *Information Exchange Packages*, below.

[XML Catalogs] are XML documents used in MPDs to ·resolve· XML schema document target namespaces to local URIs. This is especially useful when assembling an XML schema from an XML schema document set. Some validators (e.g., `Xerces`) and other tools utilize *XML catalogs* for this purpose.

> **[Definition: XML catalog document]**
>
> An XML document defined by the semantics of **[XML Catalogs]**.

The **[NIEM SSGT]** (for NIEM 3.0) automatically adds an `xml-catalog.xml` artifact to each schema document subset it generates. The NIEM 3.0 release also includes such an artifact. These ·XML catalog documents· are provided for user convenience in the case these schema document sets must be assembled into a schema.

IEPD authors must employ ·XML catalog documents· within IEPDs to facilitate validation of IEPs.

Assembling a schema or building an ·XML catalog document· from the XML schema documents of non-conformant external standards that contain `xs:include` statements can be problematic. Be aware that if an xml-catalog (resulting from processing a set of external XML schema documents) contains any two `er:uri` element entries with identical namespaces, then that xml-catalog cannot be used for XML validation. It will have to be modified to ensure that each namespace resolves to one and only one unique ·XML catalog document· `er:uri` attribute value.

In order to support ·XML schema assembly· for the purpose of ·XML schema validation·, the following rule requires that the namespaces of all XML schema documents used within an IEPD ·resolve· to a locally-unique

artifact:

[Rule 4-20] (WF-MPD) (Constraint)

An MPD with a `c:mpdClassCode` attribute value from the set `{iepd, eiem}` MUST ·resolve· each namespace it uses to a URI through one or more ·XML catalog documents·.

This rule implies that `er:NextCatalog` elements may be used within ·XML catalog documents· to connect them and control their parsing sequence. An IEPD must contain at least one ·XML catalog document· because it is the only MPD that can specify an IEP **[Definition: Information Exchange Package (IEP)]** and provide validation instructions that would require schema assembly from XML schema documents. Section 4.6, *Information Exchange Packages*, below, provides more specifics about using ·XML catalog documents· within IEPDs.

The practical reality of ·XML schema assembly· is that the process is based on several methods for identifying the correct ·XML schema documents·, and these methods may be applied in no particular order. For the purpose of this MPD Specification, the first and primary method for MPDs is the use of XML catalogs, even though NIEM schema documents often have valid URIs in `schemaLocation` attributes.

4.6. Information Exchange Packages

This section only applies to IEPDs. An IEPD is the only MPD that defines IEPs **[Definition: Information Exchange Package (IEP)]**. An IEPD does this by declaring (either implicitly or explicitly) one or more *IEP conformance targets*.

[Definition: IEP conformance target]

A class or category of IEPs which has a set of one or more validity constraints and a unique identifier. Every IEP is an instance of one or more IEP conformance targets.

This definition requires that a IEP conformance target be assigned a unique identifier, a *Conformance Target URI* that distinguishes it from all other IEP conformance targets. Similar to a URI for an MPD artifact, construct a *conformance target URI* by concatenating the IEPD's URI, the pound sign character (`#`), and a locally unique (within the IEPD) `NCName` (non-colonized name) as defined by **[W3C XML Schema Structures]**§3.3.7 NCName.

[Definition: IEP conformance target URI]

A globally unique identifier for an IEP conformance target declared in an IEPD, formed by concatenating:

1. the IEPD URI
2. the pound sign character (`#`) and
3. a locally unique `NCName` (non-colonized name).

The foregoing definition requires that an IEP conformance target class have a URI. As a result, the following rule is also required for an ·MPD catalog document·:

[Rule 4-21] (MPD-Cat) (Constraint)

A `c:IEPConformanceTarget` element MUST own a `structures:id` attribute.

An IEPD defines IEP conformance targets by explicitly declaring them within its MPD catalog. The rule above ensures that conformance targets can be referenced between IEPDs (not just within an IEPD).

The following subsections detail the concepts, artifacts, and procedures for declaring and identifying IEP conformance targets in IEPDs.

4.6.1. Schema Validation

NIEM employs the W3C XML Schema Definition (XSD) Language (**[W3C XML Schema Structures]** and **[W3C XML Schema Datatypes]**), one of several XML schema definition languages designed to define an instance XML document and enable its validation. In general, an instance XML document is valid against a particular XML schema if it obeys or conforms to the constraints imposed by that schema (**[W3C XML Schema Structures]** `§2.5 Schema-validity and documents`).

So, a NIEM IEPD is an MPD that contains a set of XML schema documents, that are assembled into an XML schema (after processing XML catalogs to ·resolve URI· values in namespace attributes owned by `xs:import` elements and similar XML Schema constructs). In turn, the resulting XML schema can be used to validate one or more instance ·XML documents· (i.e., NIEM IEPs **[Definition: Information Exchange Package (IEP)]**) for NIEM conformance.

NIEM is based on XML Schema, and so the term "schema validation" usually refers to "XML Schema validation". However, an IEPD author may also choose to include artifacts to validate with other types of schemas or rules, including but not limited to **[ISO Schematron]** and **[ISO RelaxNG]**. IEPD authors may also include artifacts for NIEM constraint schema validation, which, of course, is XML Schema validation (See Section 3.5, *Constraint Schema Documents and Document Sets*, above.

4.6.2. Declaring Validity Constraints

Explicit declaration of validity constraints is far more flexible and precise than relying on conventions that can easily be misinterpreted. The `c:IEPConformanceTarget` element within the ·MPD catalog document· can apply several common constraints by explicitly declaring the information required for a given constraint. This information may include the conformance target, context, type of validation, location of validation artifact(s), a specific tests to perform. It can also identify IEP samples known to satisfy the validity constraints.

[Rule 4-22] (MPD-Cat) (Interpretation)

An IEPD MUST explicitly declare all intended validity constraints by employing the `c:IEPConformanceTarget` element in its ·MPD catalog document·.

Appendix A, *MPD Catalog XML Schema Document*, below, provides XML elements for various validity constraints. These constraints are employed by element substitution using two abstract elements, `c:ValidityConstraintWithContext` and `c:ValidityConstraint`. Appendix A, *MPD Catalog XML Schema Document*, below, normatively specifies how this works.

Note that there may exist multiple ways to declare the same validity constraint with these elements. This rule only requires that each required validity constraint be declared once in a single form. For example, it may be possible to use either `c:HasDocumentElement` and `c:ValidToXPath` to declare the same XML document elements. However, it is only required that an IEPD author use one or the other.

Recall that this section does not apply to an MPD that is an EIEM, because it does not define an information exchange. That said, an EIEM author may have good rationales for providing validity constraints with an EIEM. However, this is not required for an EIEM.

The following subsections explain in more detail the purpose and use of each `c:IEPConformanceTarget` validity constraint.

4.6.2.1. c:ValidityConstraintWithContext

`c:ValidityConstraintWithContext` is an abstract element into which various validity constraints will be substituted, depending upon the IEPD author's intent. In the absence of an explicit context (declared by an `xPathText` attribute), validity constraint context defaults to *document* as defined in **[W3-XML-InfoSet]** §2.1 "The Document Information Item". In this default case, a specific validity constraint will substitute for `c:ValidityConstraint` which in turn, substitutes for `c:ValidityConstraintWithContext`.

4.6.2.2. c:ValidityConstraint

This is the abstract element for which a specific validity constraint will substitute if no explicit context is used (and therefore, the default *document* context applies as described in Section 4.6.2.1, *c:ValidityConstraintWithContext*, above).

4.6.2.3. c:ValidityContext

`c:ValidityContext` allows the explicit declaration of context (in `c:xPathText` attribute) for the application of a specific validity constraint. `c:ValidityContext` can contain any of the specific validity constraints that are substitutable for `c:ValidityConstraint`.

4.6.2.4. c:HasDocumentElement

`c:HasDocumentElement` is a validity constraint that identifies all intended XML document elements for an IEP conformance target, and it is directly substitutable for `c:ValidityConstraintWithContext`. This constraint ensures that an IEP artifact is rooted by one XML document element that is a member of the list of elements in its `c:qualifiedNameList` attribute. This is a common validity constraint employed by simple IEPDs that declare one or more intended XML document elements.

Note that the context of this validity constraint is always *document* as defined in **[W3-XML-InfoSet]** §2.1, "The Document Information Item". This is because it can only declare XML document elements. So, if an IEPD defines an IEP as a payload within an envelope, then `c:ValidityContext` must be used with another specific validity constraint and an explicit context declaration (`c:xPathText`).

4.6.2.5. c:ValidToXPath

`c:ValidToXPath` is a specific validity constraint whose purpose is to ensure that a condition is satisfied within an IEP. The condition is defined by an XPath expression contained in the `c:xPathText` attribute. If the XPath expression applied to a target instance ·XML document· returns a Boolean value of TRUE, then the condition is satisfied by that XML document.

This validity constraint is useful for a variety of purposes. For example, an IEPD author may require that a given `c:IEPConformanceTarget` must contain a particular element with a particular attribute whose value is an integer greater than some required minimum. An XPath expression can validate this.

`c:ValidToXPath` can also employ a simple XPath expression to validate that an IEP is rooted with an intended XML document element. However, other validity constraints can do this as well; the IEPD author may choose

the constraint representation.

Note that if `c:ValidToXPath` is used (substituted) within `c:ValidityContext` there will be two XPath expressions — the expression within `c:ValidToXPath` is the condition to validate, the other is the context (where the condition will be validated). For example, the context provided by `c:ValidityContext` might be `//my:speedingTicket`, while the `c:ValidToXPath` might require that a test for `exists(nc:DriverPerson)` be true.

This specific validity constraint as well as those that follow below can either be substituted for the `c:ValidityConstraint` or used within the `c:ValidityContext` element (i.e., substituted for its `c:ValidityConstraint` sub-element).

Note that if `c:ValidToXPath` is substituted for `c:ValidityConstraint` within the `c:ValidityContext` element, then the explicit context, the `c:xPathText` value, can imply that multiple items must be checked and each must return "true" in order for an IEP to pass the `c:ValidToXPath` constraint.

4.6.2.6. c:XMLSchemaValid

Because NIEM is based on XML Schema, then `c:XMLSchemaValid` will likely be employed by almost all IEPDs. This constraint simply ensures that an IEP artifact is schema valid to an XML schema described by and assembled with an XML Catalog. The starting XML Catalog artifact is identified by the `c:XMLCatalog` element. Note that it is possible for a starting XML catalog to link (through `er:nextCatalog`) to one or more cascading XML catalogs.

4.6.2.7. c:SchematronValid

`c:SchematronValid` is similar to `c:XMLSchemaValid`, but uses a `c:SchematronSchema` element to identify the Schematron rule file that applies to the IEP.

4.6.2.8. c:RelaxNGValid

`c:RelaxNGValid` is similar to the previous two validity constraints, but uses a `c:RelaxNGSchema` element to identify the RelaxNG schema file to which the IEP must validate.

4.6.2.9. c:ConformsToConformanceTarget

`c:ConformsToConformanceTarget` enables an IEPD author to effectively subclass and relate conformance target classes. For example, using this constraint, a given conformance target class defined by a `c:IEPConformanceTarget structures:id="A2"` can be required to also conform to another class `structures:id="A1"`. This creates an *IS-A* relationship. We say that `A2` *IS-AN* `A1`, or that `A2` *IS-A* specialization of `A1`.

Conformance target classes are related through the `c:conformanceTargetURI` attribute owned by `c:ConformsToConformanceTarget`. Recall in **[Definition: IEP conformance target URI]** that this URI is formed by the concatenation of the URI of the IEPD (`c:mpdURI`) itself, the pound character (`"#"`), and the value of the conformance class (`structures:id`) of the IEP conformance target.

4.6.2.10. c:ConformsToRule

Sometimes it is not possible to formally declare an executable validity constraint. For example, we can mandate that a data component definition must be present, must be in English, and must follow **[ISO 11179-4]**. Validating that text is present is easy, and validating that it is in English is more difficult, but validating that it obeys **[ISO 11179-4]** is essentially intractable. Thus, `c:ConformsToRule` provides an IEPD author with English text

representation as an alternative when it is not possible or not easy to define more formal validation rules or validity constraints.

4.6.3. IEP Sample XML Instance Documents

Sample IEP XML instance documents are representations of actual or example exchange data instances and can be extremely valuable artifacts in an IEPD. Sample IEPs can:

- Help an IEPD implementer to understand the original intent of the IEPD author.
- Be used by an implementer as a data point for validation of IEP conformance targets.
- Indicate or imply IEPD quality.

For these reasons, IEP samples are required for IEPDs:

[Rule 4-23] (IEP, NF-IEP) (Interpretation)

An IEPD MUST contain at least one IEP sample XML document instance that exemplifies each declared `c:IEPConformanceTarget`. If applicable, a single IEP sample MAY exemplify multiple conformance targets.

Note that this rule requires that each IEP conformance target be covered by at least one IEP sample document instance. This does not necessarily mandate a different IEP sample for each IEP conformance target. It may be possible, and is therefore acceptable, for a given IEP sample to serve as an example of one or more IEP conformance targets.

The purpose of this rule is not to provide a test for all possible IEP permutations given the schema definitions and validity constraint declarations; rather, it is to encourage IEPD authors to test their own designs, and to provide implementers with examples for additional understanding, guidance, and testing. To the extent possible, IEPD authors should strive to include sample IEPs that (1) capture real world business cases of data exchanges, and (2) exercise as many data components and validity constraints as possible. Where it makes sense, an IEPD author should strive to provide enough sample IEPs to exercise all the XML document elements (or payload root elements). If a single IEP cannot provide enough example coverage, an author may include multiple IEPs (but is not required to do so).

Each sample IEP usually illustrates a single view of the data based on a chosen set of conditions. Other views based on different conditions likely exist. An implementer will still need to review the IEPD documentation to ensure understanding of all potential conditions. Therefore, as appropriate, the author should not rely exclusively on sample IEPs to convey implementation understanding, since they will probably not account for all possible permutations.

4.7. Conformance Assertion

This section is applicable to IEPDs and EIEMs.

Independent authors build NIEM IEPDs and EIEMs from NIEM ·reference schema document sets·. Presently, a formal NIEM conformance certification process for IEPDs does not exist. Therefore, this specification recommends that an IEPD or EIEM contain an artifact that asserts some degree of NIEM conformance.

[Definition: conformance assertion]

An artifact that provides a formal or informal declaration that a NIEM IEPD or EIEM conforms to

[NIEM Conformance], **[NIEM NDR]**, and **[NIEM MPD Specification]** (this NIEM MPD Specification).

[Rule 4-24] (WF-MPD) (Constraint)

An MPD with a `c:mpdClassCode` attribute value from the set `{iepd, eiem}` MUST contain a ·conformance assertion· artifact in its ·MPD root directory·.

[Rule 4-25] (WF-MPD) (Constraint)

The file name of an MPD ·conformance assertion· MUST be of the form `conformance-assertion.*`.

A ·conformance assertion· presents information to increase the level of confidence that an IEPD or EIEM was checked for NIEM conformance and quality. It does NOT constitute a guarantee or contract. In fact, a conformance assertion can be self-asserted.

In the absence of a formal NIEM certification process, both weak and strong conformance assertions will exist. A IEPD/EIEM user or ninimplementer (who is not the author) must decide his/her level of confidence in the assertion. A self-signed artifact that simply claims an IEPD is NIEM-conformant may be considered weak. On the other hand, a stronger self-assertion could provide information that may include (but is not limited to):

- Date of assertion.
- Assertion of NIEM conformance.
- Author (name and/or organization, or sponsoring entity; indication of NIEM and XML background or experience).
- Certifier (could be author or another person/organization).

- Details of conformance verification:

    - How, what, and/or who? (e.g., automatic checks, manual checks, other reviews?)
    - Tool(s) employed? (e.g., tool, version, how used, on what, etc.)
    - Results? (e.g., issues, pass/fails, warnings, confirmations, etc.)

Inclusion of a conformance assertion made by a reputable, independent, trusted entity (person or organization) would likely increase confidence in conformance. Another strong case can be made by supplementing a conformance assertion with a formal conformance test report or similar artifact.

In the future, as NIEM procedures and tools advance, a conformance or quality report and a corresponding certificate may become required artifacts. A tool might check conformance and issue the report and certificate together as a digitally signed and hashed artifact that reports conformance, and proves both author and IEPD identity (i.e., that the IEPD is an unaltered copy of the original). For now, inclusion of a conformance assertion in an IEPD or EIEM is at the discretion of the author or sponsor.

5. Conformance Targets

The conformance targets that follow do not equate to levels of conformance, although it may be possible to sequence them as such. These conformance targets represent sets of rules. Each set can be applied to its respective conformance target independently of other rule sets.

5.1. Well Formed MPD (WF-MPD)

[Definition: well-formed MPD]

A valid **[PKZIP]** archive file that adheres to the rules defined for the well-formed MPD ·conformance target· in this MPD specification.

All MPDs use URIs to identify, find, or acquire artifacts and other resources. As such, the following definition for resolving URIs will be useful to the validation rules to be defined.

[Definition: resolve URI]

A function (or an action) that takes a URI string of the form `xs:anyURI` and returns the resource it identifies. If the resource is local and does not exist, then this function fails. If a resource is remote (e.g., a URL on the Internet or a URN of an unknown location), then this function (or action) may require human intervention to determine if the resource exists (pass) or not (fail).

[NIEM NDR], `§12.3 Reference Elements` defines a ·reference element· as follows:

[Definition: reference element]

An XML element that refers to its value by a reference attribute instead of carrying it as content.

The ·MPD catalog document· reuses NIEM Core and so it conforms to NIEM. This means that an author may use one or more ·reference elements· from various locations to refer to a single content bearing instance of the same element (with a unique `structures:id`). The definition of ·resolve URI· and the conformance target rules that follow assume content bearing elements. Therefore, if a rule applies to a conformance target with a URI attribute whose owning element is in ·reference element· form, then URI resolution will be applied at the site of the content-bearing element form referenced.

The following rules apply to a ·well-formed MPD·:

[Rule 5-1] (WF-MPD) (Constraint)

An MPD contains one and only one `mpd-catalog.xml` artifact in its ·root directory·.

[Rule 5-2] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute MUST ·resolve· to a resource.

[Rule 5-3] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:XMLCatalog` element MUST ·resolve· to an ·XML catalog document·.

[Rule 5-4] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:MPDChangeLog` element MUST ·resolve· to a ·change log·.

[Rule 5-5] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ReadMe` element MUST ·resolve· to a ·read-me artifact·.

[Rule 5-6] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:IEPSampleXMLDocument` element MUST ·resolve· to an ·XML document·.

[Rule 5-7] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:BusinessRulesArtifact` element MUST ·resolve· to a ·business rule schema· or ·business rules· artifact.

[Rule 5-8] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ExternalSchemaDocument` element MUST ·resolve· to an ·XML schema document·.

[Rule 5-9] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ReferenceSchemaDocument` element MUST ·resolve· to a NIEM ·reference schema document·.

[Rule 5-10] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ExtensionSchemaDocument` element MUST ·resolve· to a NIEM ·extension schema document·.

[Rule 5-11] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:SubsetSchemaDocument` element MUST ·resolve· to a NIEM ·subset schema document·.

[Rule 5-12] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:Wantlist` element MUST ·resolve· to a ·NIEM wantlist· XML document.

[Rule 5-13] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:SchematronSchema` element MUST ·resolve· to a ·Schematron schema·.

[Rule 5-14] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:RelaxNGSchema` element MUST ·resolve· to a ·RelaxNG schema·.

[Rule 5-15] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:SchemaDocumentSet` element MUST ·resolve· to an ·XML schema document· set.

[Rule 5-16] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ConstraintSchemaDocumentSet` element MUST ·resolve· to a NIEM ·constraint schema document set·.

[Rule 5-17] (WF-MPD) (Constraint)

Within the ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:ReferenceSchemaDocumentSet` element MUST ·resolve· to a NIEM ·reference schema document set·.

[Rule 5-18] (WF-MPD) (Constraint)

Within an ·MPD catalog document· the value of a `c:pathURI` attribute owned by a `c:RequiredFile` element MUST ·resolve· to a resource.

5.1.1. XML Catalog

[Rule 5-19] (XML-Cat) (Constraint)

Within an ·XML catalog document· the value of a `uri` attribute owned by a `er:uri` element MUST ·resolve· to a resource.

[Rule 5-20] (XML-Cat) (Constraint)

Within an ·XML catalog document·, given an ·XML schema document· resolved by the value of a `uri` attribute owned by a `er:uri` element, the ·XML schema document· target namespace MUST equal the value of the `name` (a namespace string) attribute owned by the `er:uri` element.

5.2. IEP

[Rule 5-21] (IEP) (Constraint)

Within an ·MPD catalog document· with a `c:xPathText` attribute owned by a `c:ValidToXPath` element, a candidate IEP is a valid IEP, ONLY IF the value of `c:ValidToXPath` applied to the candidate IEP (an ·XML document·) has an effective Boolean value (EBV) equal to `true`. EBV is defined by **[W3C XPath 2.0]** § 2.4.3 “Effective Boolean Value”.

[Rule 5-22] (IEP) (Constraint)

Within an ·MPD catalog document· with a `c:qualifiedNameList` attribute owned by a `c:HasDocumentElement` element, a candidate IEP is a valid IEP, ONLY IF its XML document element's name is a member of that list of *QNames*.

[Rule 5-23] (IEP) (Constraint)

Within an ·MPD catalog document· with a `c:pathURI` attribute owned by a `c:IEPSampleXMLDocument`, the artifact resolved by the value of `c:pathURI` MUST be valid for the validity constraints of the `c:IEPConformanceTarget` parent of `c:IEPSampleXMLDocument`.

5.3. Full NIEM IEP (FN-IEP)

[Definition: full NIEM IEP]

An XML document with an XML document element that is defined by a NIEM reference schema or NIEM extension schema. (Abbreviated as "FN-IEP")

The following rules apply to a ·full NIEM IEP·:

[Rule 5-24] (FN-IEP) (Constraint)

Within an ·MPD catalog document· the value of a `c:externalURI` attribute owned by an element that is a `c:FileType` (or a type derived from it) MUST ·resolve· to a resource.

[Rule 5-25] (FN-IEP) (Constraint)

Within an ·MPD catalog document· the value of a `c:externalURI` attribute owned by an element that is a `c:FileSetType` (or a type derived from it) MUST ·resolve· to a resource.

[Rule 5-26] (FN-IEP) (Constraint)

Within an ·MPD catalog document· the value of a `c:resourceURI` attribute owned by a `c:Relationship` element MUST ·resolve· to a resource.

5.4. NIEM IEPD (N-IEPD)

[Definition: NIEM IEPD]

A special kind of MPD that establishes at least one conformance target for a class of IEPs (by employing one or more `c:IEPConformanceTarget` elements. Furthermore, satisfies all rules and mandatory requirements for:

- **[NIEM MPD Specification]** (i.e., This MPD Specification)
- **[NIEM NDR]**
- **[NIEM Conformance]**

[TBD]

6. Optional MPD Artifacts

Aside from the required artifacts, MPD content is relatively flexible. A variety of other optional documentation files may be incorporated into an MPD. When applicable, these may include (but are not limited to) files that describe or explain:

- Implementation details (hardware, software, configuration, etc.)
- Use of multiple root elements
- Use of multiple subsets or mixed releases
- How to use/reuse an MPD for various purposes (such as Web Services)
- Rationales and/or business purposes

In addition to documentation artifacts, a variety of other optional files can be added to an MPD to facilitate tool support and make reuse, adaptation, and/or implementation easier. These are often files that are inputs to or outputs from software tools. Examples include content diagrams, content models in tool-specific formats, and business rules (either formal or informal representations).

An MPD author may include any files believed to be useful to understand, implement, reuse, and/or adapt an MPD.

An MPD of relatively simple content and scope may only need to contain the minimum mandatory artifacts required by this specification in order to understand and implement it. (See Appendix D, *Common MPD Artifacts*, below, for a listing of the mandatory and common optional artifacts for each type of MPD.)

Files vary widely in format and are often specific to the tools an author uses to parse, consume, or output them. Therefore, if tool-specific files are included in an MPD, it is also a good practice to include copies of those files in formats that display with standard Web browsers or other cost-free, publicly available viewing tools (e.g., ASCII text, PDF, CSV, HTML, JPG, GIF, PNG). This guidance is intended to encourage and facilitate maximal sharing and distribution of MPDs; it does not prohibit and is not intended to discourage the inclusion of other file formats.

In particular, this specification does not discourage use of Microsoft file formats for documentation and other optional artifacts. Microsoft Office products are in common use, and free viewers are available for many of them (See `http://office.microsoft.com/en-us/downloads/office-online-file-converters-and-viewers-HA001044981.aspx`).

6.1. NIEM Wantlist

A NIEM schema document subset is often associated with a NIEM *wantlist*. A *wantlist* is an abbreviated XML representation of a NIEM schema document subset, and identifies only the data components a user selected (as requirements) to build a schema document subset. To reconstruct the complete schema document subset there are usually a number of additional data components that the user selections depend upon. These must be computed from the appropriate NIEM reference model and added to reconstruct the complete schema document subset. For example, a user may select `nc:Person` for the subset. In this case, the wantlist will only contain that component, but the associated full subset must contain both `nc:Person` and `nc:PersonType`. A software tool that understands how to process NIEM wantlists and schema document subsets (such as the NIEM Schema Subset Generator Tool **[NIEM SSGT]**) can rebuild an accurate schema document subset from a wantlist (and the reverse).

> **[Definition: NIEM wantlist]**
>
> An XML document that represents a complete NIEM schema document subset.

A NIEM wantlist identifies the data component requirements declared by the subset author; it does not identify the data component dependencies required to reconstitute the complete subset. The complete subset can be computed with the reference schema document set from which the subset was derived.

A wantlist is always associated with a schema document subset. A wantlist may also be associated with a constraint schema document set, because constraint schema documents are often built from a schema document subset. For a simple IEPD, it can sometimes be trivial to identify a single schema document subset. However, this

MPD Specification does not prohibit building complex IEPDs that contain schema document sets supported by multiple schema document subsets and associated wantlists. As with other complex cases, the IEPD author is responsible to clearly document the associations between wantlists and schema document sets. In order to maintain a minimal degree of consistency for placement of a wantlist within an IEPD or EIEM:

[Rule 6-1] (WF-MPD) (Constraint)

If present, a NIEM wantlist MUST reside within the root of the MPD (`niem`) subdirectory that groups and defines its corresponding subset schema document set.

6.2. Business Rules

For simplicity and consistency, NIEM employs a profile of the XML Schema language **[W3C XML Schema Structures]**, **[W3C XML Schema Datatypes]**. Thus, some constraints on NIEM XML documents cannot be enforced by NIEM. ·Constraint schema documents· provide a convenient technique for enforcing some additional constraints. However, even the full XML Schema language cannot validate and enforce all possible constraints that may required on an XML document.

That said, NIEM allows (even encourages) the use of formal or informal *business rules* to supplement MPDs (in particular IEPDs).

[Definition: business rules]

Formal or informal statements that describe business policy or procedure, and in doing so define or constrain some aspect of a process or procedure in order to impose control.

·Business rules· may be represented as informal English statements, or as formally coded machine-readable and processible statements. For example, an IEPD may use a Schematron schema **[ISO Schematron]**, a RelaxNG schema **[ISO RelaxNG]**, or any other formal representation for ·business rules·.

[Definition: business rule schema]

An artifact that contains ·business rules· in a formal representation language with the intent to automatically process them on an XML document to enforce business constraints.

[Definition: Schematron schema]

A ·business rule schema· that adheres to **[ISO Schematron]**.

[Definition: RelaxNG schema]

A ·business rule schema· that adheres to **[ISO RelaxNG]**.

7. Organization, Packaging, and Other Criteria

An MPD is a logical set of electronic files aggregated and organized to fulfill a specific purpose in NIEM.

Directory organization and packaging of an MPD should be designed around major themes in NIEM: reuse, sharing, interoperability, and efficiency.

This rule is also applicable to all MPDs:

[Rule 7-1] (WF-MPD) (Constraint)

An MPD is packaged as a single compressed archive of files that represents a sub-tree of a file system in standard **[PKZIP]** format. This archive MUST preserve and store the logical directory structure intended by its author.

MPD NIEM schema artifacts must be valid for both XML Schema and NIEM:

[Rule 7-2] (WF-MPD) (Constraint)

Within an MPD archive, all XSD and XML artifacts MUST be valid against and follow all rules for their respective **[NIEM NDR]** conformance targets; this includes being well-formed and valid XML Schema documents.

NIEM releases, core updates, and domain updates maintain a relatively consistent directory organization **[NIEM Domain Update Specification]**. But there are many ways to organize IEPD and EIEM directories that may depend on a number of factors including (not limited to) business purpose and complexity. For this reason, strict rules for IEPD and EIEM directory structure are difficult to establish. Therefore, IEPD and EIEM authors may create their own logical directory structures subject to the rules of this section.

[Definition: MPD root directory]

The top level file directory relative to all MPD artifacts and subdirectories.

[Rule 7-3] (WF-MPD) (Constraint)

An MPD archive MUST uncompress (unzip) to one and only one ·MPD root directory·.

The foregoing rule ensures that:

- Unpacking an MPD archive will not scatter its contents on a storage device.
- A common starting point always exists to explore or use any MPD.
- mpd-catalog and change log artifacts will always be found in the ·MPD root directory·.

7.1. Artifact Sets

This specification defines different ways to group MPD artifacts into ·sets·. In general, all sets are established through an ·XML catalog document· or the ·MPD catalog document·.

Section 4.5, *XML Catalogs*, above, describes how NIEM employs an ·XML catalog document· to assemble an ·XML Schema· from ·XML schema documents·. This is one important method for grouping XML schema

documents into sets. In fact, this is the required standard technique for grouping schema documents for assembling them into a schema. For user convenience, this method is now used in each ·release·, ·core update·, and ·domain update·. Note also that this method is applicable to all the various classes of NIEM XML schema documents (reference, subset, extension, constraint, and external).

Another reason for grouping artifacts into sets is a common need for humans to review, identify, and navigate the artifacts of an IEPD (particularly, if it's complicated). An ·XML catalog document· has a relatively focused purpose, to select and assemble a set of XML schema documents into an ·XML Schema·. It is not intended to index artifacts in general (other than XML schema documents to assigned target namespaces). So, it does not classify or describe the artifacts it identifies.

On the other hand, the ·MPD catalog document· is designed to record, index, classify, and describe (as needed) any or all MPD artifacts (not just schema documents). The MPD catalog provides a more flexible method for grouping all artifacts.

The MPD catalog schema Appendix A, *MPD Catalog XML Schema Document*, below, defines a large set of common artifact classifiers and a smaller set of artifact set classifiers. In summary, per Appendix A, *MPD Catalog XML Schema Document*, below, define sets by substituting the appropriate artifact classifier (of type `c:FileType`) into the abstract element `c:Artifact`, within the appropriate artifact set classifier (of type `c:FileSetType`). Use the most specific classifiers available for your artifacts and artifact sets. Otherwise, as needed, use generic `c:File` and `c:FileSet` classifiers with `nc:DescriptionText`.

Note that the `c:pathURI` value for an artifact is its operating system relative directory path name and file name. The `c:pathURI` value for an artifact set is its operating system relative directory path name.

Another way the MPD catalog schema can group artifacts is through the use of the `c:RequiredFile` element. Use this construct to signify there are strong dependencies among artifacts. For example, documentation may be prepared as a set of hyperlinked HTML files. These HTML files may also incorporate separate GIF or JPG images. Regardless of file location within the MPD, these files depend on one another through hyperlinks. As a result, they tend operate as a single artifact; removal of a file will cause one or more broken links within the set. This set of artifacts should be grouped using `c:RequiredFile`. If the set does not have a root HTML document (i.e., the set can be entered from any file in the set), then create an index HTML document and use it as the root of the set (i.e., the index is the value of the `c:File` element while all others are values for `c:RequiredFile` child elements).

7.2. MPD File Name Syntax

This section is applicable to all MPDs. However, it is only normative for releases, core updates, and domain updates. For IEPDs and EIEMs, this section describes non-normative guidance that is highly recommended, but not mandatory. Additional non-normative guidance that relates to directory naming for IEPDs and EIEMs is in Appendix F, *Guidance for IEPD Directories (non-normative)*, below.

The MPD Specification is intended to help facilitate tool support for processing MPDs. Providing tools and search mechanisms basic information about an MPD as early as possible will help reduce processing time and complexity. So, if the MPD name, version, and class can be identified from its file name, then a tool would not have to open the archive and parse the MPD catalog to determine such. Of course, ultimately, to do anything useful, a tool will have to open the MPD archive. However, standard file name syntax allows a tool to search through a set of MPDs to find a particular MPD name, version, or class without having to open each. File name consistency can also make it easier to scan and identify MPDs in a long list sorted by file name.

The following rule applies to releases, core updates, and domain updates:

[Rule 7-4] (WF-MPD) (Constraint)

Given an MPD whose `c:mpdClassCode` value is in the set `{rel, cu, du}`, its file name MUST adhere to the syntax defined by the regular expression:

```
mpd-filename ::= name '-' version '.' class '.zip'
Where:
        name     ::= alphanum ((alphanum | special)* alphanum)?
        alphanum ::= [a-z0-9]
        special  ::= '.' | '-' | '_'
        version  ::= digit+ ('.' digit+)* (status digit+)?
        digit    ::= [0-9]
        status   ::= 'alpha' | 'beta' | 'rc' | 'rev'
        class    ::= 'rel' | 'cu' | 'du' | 'iepd' | 'eiem'
```

The `status` values are as defined in Rule 4-10.

The `class` values are defined as follows:

- `rel` = release
- `cu` = core update
- `du` = domain update
- `iepd` = information exchange package documentation
- `eiem` = enterprise information exchange model

The regular expression notation in the rule above is from **[W3-XML]** `#sec-notation`.

Alphabetic characters are lower case to reduce complications across various file systems.

An example of an intermediate release file name that follows these rules is: `niem-3.0beta1.rel.zip`

MPD file names for releases, core updates, and domain updates also adhere to this rule:

[Rule 7-5] (WF-MPD) (Constraint)

Given an MPD with a `c:mpdClassCode` attribute value from the set `{rel, cu, du}`, its file name substrings for `name`, `version`, and `class` (as defined in Rule 7-4 MUST exactly match its MPD catalog values for `c:MPD` attributes `c:mpdName`, `c:mpdVersionID`, and `c:mpdClassCode` respectively.

And finally, in HTTP-based Web Services environments, the MIME type designation of a MPD archive is important to facilitate processing by service consumers.

[Rule 7-6] (WF-MPD) (Constraint)

When represented on the Internet, an MPD archive SHOULD use the following MIME Type:

```
application/zip+[class]
        where [class] is one member from the value set
        {rel, cu, du, iepd, eiem}.
```

Use of the generic Zip MIME type `application/zip` is allowed, but discouraged. No other MIME types are allowed when representing MPD archives.

7.3. Artifact Links to Other Resources

It is important to understand that the URI scheme defined in Section 4.2.3, *URI Scheme for MPD Artifacts*, above, can only be used to identify relationships among and provide source links to external schemas being reused. It is not sufficient to allow references or links to such schemas stand in for a physical copy. Thus, all schema artifacts necessary to define, validate, and use an MPD must be physically present within that MPD. In accordance with the **[NIEM NDR]**, if MPD schemas are moved to an operational environment for implementation, validation, or other purposes, then absolute references may replace relative path references when needed. When absolute references to Internet resources are required:

[Rule 7-7] (WF-MPD) (Constraint)

Absolute references to Internet resources MUST use a well-known transfer protocol (http, https, ftp, ftps) and MUST ·resolve· (If applicable, documentation SHOULD describe how to ·resolve· with security, account, and/or password issues).

Releases, core updates, and domain updates must adhere to packaging rules primarily to enable development tools to process them consistently and efficiently. The NIEM PMO controls the format and documentation for these MPDs and publishes them at `http://release.niem.gov/niem/`. However, many different organizations author IEPDs and EIEMs. As such, they may be distributed, published in repositories (possibly to a limited community), and reused by others. Furthermore, EIEMs are the basis for families of IEPDs. Therefore, it is important that both of these MPD classes are well documented for understanding and use. An IEPD that has been derived from an EIEM should identify that relationship through its MPD catalog.

[Rule 7-8] (N-IEPD) (Constraint)

An ·MPD catalog document· for an IEPD that is a member of a family of IEPDs derived from a given EIEM, MUST have a `c:MPDInformation/c:Relationship` element that owns a `c:RelationshipCode="derived_from"` attribute and a `c:resourceURI` attribute with a URI value that ·resolves· to an EIEM resource.

7.4. IEPD Completeness

Since an IEPD defines an information exchange and is often implemented by persons other than the original author, it is important to ensure that they are relatively complete and provide all artifacts needed to use the IEPD.

[Rule 7-9] (N-IEPD) (Constraint)

An IEPD SHOULD contain all artifacts needed to understand it and facilitate its correct implementation.

The rule above means that an IEPD implementer should not be forced to search for or track down specialized schema documents, documentation, or other artifacts required to validate and implement exchanges defined by an IEPD. Specialized artifacts refer to those designed and built by an IEPD author, not artifacts that are standards and publicly available to all implementers. For example, this rule does not imply that an IEPD should contain a schema document that defines the XML schema component vocabulary identified by the namespace name `http://www.w3.org/2001/XMLSchema` (i.e., XS), or `http://www.w3.org/2001/XMLSchema-instance` (i.e., XSI). All schema processors have appropriate declarations for these built in. Likewise, an IEPD is not required to contain `mpd-catalog-3.0.xsd` or the standard NIEM subset that supports it.

On the other hand, an IEPD whose author has extended the MPD catalog schema is clearly required to contain the catalog extension schema document, since this is a specialized customization created by the author. If a different NIEM schema subset is also used, then the IEPD must also contain its superset (i.e., a complete subset that incorporates both the original subset with additional NIEM components used to extend the catalog schema document; see Rule 4-9, above.)

The rationale for "SHOULD" in Rule 7-9, above, relates to issues of security. Although NIEM is generally public, some IEPDs (and even other MPDs) may contain XML tags that provide more semantics or structure than a domain is willing to expose. In such cases, it may be necessary to simply refer to schema documents that are required for validition and implementation, instead of circulating them within a public IEPD. Implementers would then be expected to know how and where to obtain the required documents.

The **[NIEM NDR]** explains how NIEM employs adapter types to encapsulate and use other standards (e.g., geospatial and emergency management standards) in their native forms that are not NIEM-conformant. Other standards may use `xs:import` without requiring `schemaLocation` attributes (instead, relying only on the namespace value). These standards may also use `xs:include`. This XML Schema construct is disallowed by NIEM. When standards external to NIEM are required within MPDs, the following rule applies:

[Rule 7-10] (WF-MPD) (Constraint)

Within an MPD, if non-conformant external schema documents are used, then any references from these schema documents to other schema documents and/or namespaces MUST ·resolve· to local URIs. `schemaLocation` attributes or XML catalogs can be used to ensure resolution.

For the case of non-NIEM-conformant schemas, this rule ensures that all schemas (or corresponding artifacts and namespaces) from external standards required for definition, validation, and use of the MPD are present within the archive.

XML schemas are the heart of MPDs since they formally specify normative structure and semantics for data components. However, in general, an MPD is a closed set of artifacts. This means that all hyperlink references within artifacts should ·resolve· to the appropriate artifact.

[Rule 7-11] (WF-MPD) (Constraint)

Within any artifact of an MPD archive, any direct reference to another resource (i.e., another artifact such as an image, schema, stylesheet, etc.) that is required to process or display an artifact SHOULD exist within the archive at the location specified by that reference.

This means that MPD artifacts, including documentation artifacts, should be complete. For example, if an HTML document within an MPD contains a hyperlink reference (`href`) to an artifact that is part of or used by the MPD, then the file associated with that hyperlink should be present in the MPD; likewise for a sourced (`src`) image. Authors should exercise good judgment with this rule. For example, it does not require an MPD to contain copies of all cited documents from a table of references if it contains hyperlinks to those documents. The key operating words in this rule are: "another resource is required to process or display an artifact SHOULD exist within the archive."

In some cases, it may not be possible to include all artifacts, even schemas, in an MPD without violating laws, regulations, or policies. For example, an IEPD may require use of a schema document that is not publicly accessible; it might be classified or controlled unclassified information (CUI). This is a valid reason for exception to Rule 7-11, above. If the IEPD is placed in the public domain, the author should omit the non-public schema

document, and if appropriate, document the omission, and explain where and/or how the missing schema document can be obtained.

7.5. Duplication of Artifacts

Within an MPD, the replication of files or entire file sets should be avoided. However, replication is allowed if a reasonable rationale exists. In some cases, file replication may make it easier to use, validate, implement, or automatically process an MPD. For example, multiple subsets and/or constraint sets may overlap in many identical schema documents. Yet, allowing this duplication may be easier or necessary to accommodate a validation tool, rather than removing duplicate schema documents, and forcing the tool to search for them. Whenever possible, use XML catalogs to coordinate schema assembly.

Appendix A. MPD Catalog XML Schema Document

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema
    ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and-
design-rules/3.0/#ExtensionSchemaDocument"
    targetNamespace="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
    version="3.0"
    xmlns:appinfo="http://release.niem.gov/niem/appinfo/3.0/"
    xmlns:c="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
    xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
    xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
    xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
    xmlns:term="http://release.niem.gov/niem/localTerminology/3.0/"
    xmlns:niem-xs="http://release.niem.gov/niem/proxy/xsd/3.0/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:import namespace="http://release.niem.gov/niem/structures/3.0/"/>
    <xs:import namespace="http://release.niem.gov/niem/niem-core/3.0/"/>
    <xs:import namespace="http://release.niem.gov/niem/proxy/xsd/3.0/"/>

    <xs:annotation>

      <xs:documentation>
          Model Package Description (MPD) Catalog.
          This schema defines an mpd-catalog.xml artifact
          for NIEM Model Package Descriptions (MPD):
             NIEM releases, Core updates, domain updates,
             NIEM Information Exchange Package Documentation (IEPD),
             and NIEM Enterprise Information Exchange Models (EIEM).
          The purpose of this schema is to facilitate consistent declaration
          of MPD content, metadata, and lineage to process, display, review,
          register, search, and discover MPDs efficiently. For IEPDs, the
          mpd-catalog provides instructions for validating IEPs to schemas.

          This XML Schema document is supported by a subset of niem-core v3.0.
          c:AuthoritativeSource should use NIEM 3.0 schema components.
      </xs:documentation>

      <xs:appinfo>
        <term:LocalTerm term="EIEM"  literal="Enterprise Information Exchange
Model"/>
        <term:LocalTerm term="EXI"   literal="Efficient XML Interchange"/>
        <term:LocalTerm term="IANA"  literal="Internet Assigned Numbers Authority"/>
        <term:LocalTerm term="ID"    literal="Identifier"/>
        <term:LocalTerm term="IEP"   literal="Information Exchange Package"
                                     definition="an instance XML document"/>
        <term:LocalTerm term="IEPD"  literal="Information Exchange Package
                                              Documentation"/>
        <term:LocalTerm term="MIME"  literal="Multipurpose Internet Mail Extension"/>
```

```
        <term:LocalTerm term="MPD"   literal="Model Package Description"/>
        <term:LocalTerm term="OASIS" literal="Organization for the Advancement
                                              of Structured Information
Standards"/>
        <term:LocalTerm term="SSGT"  literal="Schema Subset Generation Tool"/>
        <term:LocalTerm term="URI"   literal="Uniform Resource Identifier"/>
        <term:LocalTerm term="Wantlist" definition="An XML file that represents a
NIEM
                        schema document subset; used by NIEM Schema Subset Generation
                        Tool to input/output a schema document subset"/>
      </xs:appinfo>

    </xs:annotation>

    <xs:element name="Catalog" type="c:CatalogType">
      <xs:annotation>
        <xs:documentation>An MPD catalog that describes MPD artifacts and metadata.
        </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="CatalogType">
      <xs:annotation>
        <xs:documentation>A data type for an MPD catalog.</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="c:MPD"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="MPD" type="c:MPDType">
      <xs:annotation>
        <xs:documentation>A Model Package Description (MPD).</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="MPDType">
      <xs:annotation>
        <xs:documentation>A data type for an MPD.</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
           <xs:element ref="nc:DescriptionText" minOccurs="0"/>
           <xs:element ref="c:MPDInformation"   minOccurs="0"/>
           <xs:element ref="c:IEPConformanceTarget" minOccurs="0"
maxOccurs="unbounded"/>
           <xs:element ref="c:ArtifactOrArtifactSet" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="c:mpdURI"       use="required"/>
          <xs:attribute ref="c:mpdClassCode" use="required"/>
          <xs:attribute ref="c:mpdName"      use="required"/>
          <xs:attribute ref="c:mpdVersionID" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="ArtifactOrArtifactSet" abstract="true">
      <xs:annotation>
              <xs:documentation>
                 A concept for a file or file set in an MPD.
```

```
                </xs:documentation>
      </xs:annotation>
    </xs:element>

<!-- File artifact classifiers for a table of contents ========================== --
>

    <xs:element name="File" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                 A generic electronic file artifact in an MPD; a file stored on a
computer system.
              </xs:documentation>
      </xs:annotation>
      </xs:element>

    <xs:complexType name="FileType">
      <xs:annotation>
        <xs:documentation>A data type for an MPD file artifact.</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="c:RequiredFile"     minOccurs="0"
maxOccurs="unbounded"/>
          <xs:element ref="nc:DescriptionText" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="c:pathURI"           use="required"/>
          <xs:attribute ref="c:mimeMediaTypeText" use="optional"/>
          <xs:attribute ref="c:externalURI"       use="optional"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="XMLCatalog" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                 An MPD artifact that is an OASIS XML catalog.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="MPDChangeLog" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                 An MPD artifact that contains a record of the MPD changes.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ReadMe" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                 An MPD read-me artifact.
         </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="IEPSampleXMLDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
```

```
                An example MPD instance XML document or IEP artifact.
             </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="BusinessRulesArtifact" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
             <xs:documentation>
                An MPD artifact that contains business rules
                and constraints on exchange content.
             </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="XMLSchemaDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
             <xs:documentation>
                An MPD artifact that is a an XML schema document (i.e., an XSD that
                is not necessarily a NIEM subset, extension, or reference schema).
             </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ExternalSchemaDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
             <xs:documentation>
                An MPD artifact that is a schema document external to NIEM.
             </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ReferenceSchemaDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
             <xs:documentation>
                An MPD artifact that is a NIEM reference schema document.
             </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ExtensionSchemaDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>An MPD artifact that is a NIEM extension schema document.
        </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="SubsetSchemaDocument" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>An MPD artifact that is a subset schema document.
        </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="EXIXMLSchema" type="c:XMLSchemaType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
             <xs:documentation>
                     An XML Schema to be used for EXI serialization of an IEP Class.
             </xs:documentation>
      </xs:annotation>
```

```
    </xs:element>

    <xs:element name="Wantlist" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>An MPD artifact that represents a NIEM schema subset
        and is used as an import or export for the NIEM SSGT.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ConformanceAssertion" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>An MPD artifact that is a signed declaration
        that a NIEM IEPD or EIEM is NIEM-conformant.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ConformanceReport" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>
           An MPD artifact auto-generated by a NIEM-aware software tool that checks
           NIEM conformance and/or quality and renders a detailed report of results.
        </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="SchematronSchema" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>A Schematron schema document.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="RelaxNGSchema" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
        <xs:documentation>A RelaxNG schema.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="Documentation" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
         <xs:documentation>
                An MPD artifact that is a form of explanatory documentation.
         </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ApplicationInfo" type="c:FileType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                      An MPD artifact that is used by a software tool (e.g., import,
export, input, output, etc.).
              </xs:documentation>
      </xs:annotation>
    </xs:element>


<!-- For declaring file dependencies ============================================= --
>

    <xs:element name="RequiredFile" type="c:FileType">
```

```
      <xs:annotation>
              <xs:documentation>
                      An MPD file artifact that another artifact depends on and
should not be separated from.
              </xs:documentation>
      </xs:annotation>
    </xs:element>


<!-- Artifact Set Classifiers ================================================== --
>

    <xs:element name="FileSet" type="c:FileSetType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                      A generic MPD artifact set; used to group artifacts that are
not accounted for by other set classifiers.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="FileSetType">
      <xs:annotation>
        <xs:documentation>A data type for an MPD file artifact set.
</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:ArtifactOrArtifactSet" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="c:pathURI"     use="required"/>
          <xs:attribute ref="c:externalURI" use="optional"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="SchemaDocumentSet"
              type="c:SchemaDocumentSetType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                      An MPD artifact set that may include subset schema documents,
extension and external schema documents, and other supporting artifacts.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ConstraintSchemaDocumentSet"
             type="c:SchemaDocumentSetType"
substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                      An MPD artifact set of constraint schema documents and other
supporting artifacts.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

   <xs:complexType name="SchemaDocumentSetType">
      <xs:annotation>
              <xs:documentation>
                      A data type for an MPD artifact set that may include subset
```

```
schema documents, extension schema documents, and external schema documents or
constraint schema documents.
            </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:restriction base="c:FileSetType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText"  minOccurs="0"/>
            <xs:element ref="c:XMLCatalog"/>
            <xs:element ref="c:Wantlist"          minOccurs="0"/>
            <xs:element ref="c:XMLSchemaDocument" minOccurs="0"
                                                  maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="ReferenceSchemaDocumentSet"
type="c:ReferenceSchemaDocumentSetType" substitutionGroup="c:ArtifactOrArtifactSet">
      <xs:annotation>
              <xs:documentation>
                      An MPD artifact set of reference schema documents and other
supporting artifacts.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="ReferenceSchemaDocumentSetType">
      <xs:annotation>
              <xs:documentation>
                      A data type for an MPD artifact set that is a reference schema
document set.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:restriction base="c:FileSetType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:ReferenceSchemaDocument" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>


<!-- Primitives ================================================================== --
>

    <xs:attribute name="mpdURI" type="xs:anyURI">
      <xs:annotation>
              <xs:documentation>
                      A globally unique identifier (URI) for an MPD.
              </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="mpdName" type="c:MPDNameSimpleType">
      <xs:annotation>
        <xs:documentation>A descriptive label or title for an MPD.</xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:simpleType name="MPDNameSimpleType">
      <xs:annotation>
                <xs:documentation>
                        A data type for an MPD name, label, or title.
```

```
                </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:token">
        <xs:pattern value="[a-z]([-_]?[a-z0-9]+)*"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:attribute name="mpdVersionID" type="c:MPDVersionIDSimpleType">
      <xs:annotation>
                <xs:documentation>
                        An identifier that distinguishes releases of a given MPD.
                </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:simpleType name="MPDVersionIDSimpleType">
      <xs:annotation>
                <xs:documentation>
                        A data type for an identifier that distinguishes releases of
a given MPD.
                </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:token">
        <xs:pattern value="[0-9]+(\.[0-9]+)*((alpha|beta|rc|rev)[0-9]+)?"/>
      </xs:restriction>
    </xs:simpleType>

<!-- ========================================================================== -->

    <xs:attribute name="mpdClassCode" type="c:MPDClassListSimpleType">
      <xs:annotation>
                <xs:documentation>
                        A classification for an MPD; values drawn from the set {rel,
cu, du, iepd, eiem}.
                </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:simpleType name="MPDClassListSimpleType">
        <xs:annotation>
                <xs:documentation>
                        A data type that ensures at least one class is identified for
an MPD.
                </xs:documentation>
        </xs:annotation>
        <xs:restriction base="c:ClassListSimpleType">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ClassListSimpleType">
        <xs:annotation>
                <xs:documentation>
                        A data type for one or more codes or URIs for MPD classes.
                </xs:documentation>
        </xs:annotation>
        <xs:list itemType="c:ClassUnionSimpleType"/>
    </xs:simpleType>

    <xs:simpleType name="ClassUnionSimpleType">
        <xs:annotation>
                <xs:documentation>
                        A data type for one of five MPD classes or a new class
identified by URI.
                </xs:documentation>
        </xs:annotation>
```

```
        <xs:union memberTypes="c:MPDClassCodeSimpleType xs:anyURI"/>
    </xs:simpleType>

    <xs:simpleType name="MPDClassCodeSimpleType">
      <xs:annotation>
         <xs:documentation>A data type for the classification of an MPD.
</xs:documentation>
      </xs:annotation>
        <xs:restriction base="xs:token">
            <xs:enumeration value="rel">
                <xs:annotation>
                  <xs:documentation>release</xs:documentation>
                </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="cu">
                <xs:annotation>
                  <xs:documentation>core update</xs:documentation>
                </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="du">
                <xs:annotation>
                  <xs:documentation>domain update</xs:documentation>
                </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="iepd">
                <xs:annotation>
                  <xs:documentation>information exchange package documentation
                  </xs:documentation>
                </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="eiem">
                <xs:annotation>
                  <xs:documentation>enterprise information exchange
model</xs:documentation>
                </xs:annotation>
            </xs:enumeration>
        </xs:restriction>
    </xs:simpleType>

<!-- ================================================================================ -->

    <xs:attribute name="pathURI" type="xs:anyURI">
      <xs:annotation>
              <xs:documentation>
                      A URI for the pathname of a local artifact relative to the MPD
root directory.
              </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="externalURI" type="xs:anyURI">
      <xs:annotation>
              <xs:documentation>
                      A globally unique identifier (URI) for an artifact in another
MPD that is reused by this MPD.
              </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="mimeMediaTypeText" type="xs:string">
      <xs:annotation>
              <xs:documentation>
                      A classification for an MPD file artifact from the IANA MIME
media classes: http://www.iana.org/assignments/media-types.
              </xs:documentation>
      </xs:annotation>
```

```
    </xs:attribute>

    <xs:complexType name="RelationshipType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a reference to another MPD related to this MPD.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="c:relationshipCode" use="required"/>
          <xs:attribute ref="c:resourceURI"      use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:attribute name="resourceURI" type="xs:anyURI">
      <xs:annotation>
              <xs:documentation>
                      A globally unique identifier (URI) for another MPD or document
to which this MPD relates.
              </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="relationshipCode" type="c:RelationshipCodeSimpleType">
      <xs:annotation>
              <xs:documentation>
                      A classification or reason for the connectedness between this
MPD and the resource referenced in resourceURI.
              </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:simpleType name="RelationshipCodeSimpleType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a classification of the relationship between
MPDs.
              </xs:documentation>
      </xs:annotation>
        <xs:restriction base="xs:token">
            <xs:enumeration value="version_of">
              <xs:annotation>
                      <xs:documentation>
                              A relationshipCode value for indicating that this MPD
is a different version of the MPD referenced in resourceURI.  This code value is only
needed in cases where significant name changes might obscure the relationship to the
previous version.  For example, NIEM Justice 4.1 is a version of GJXDM 3.0.3.
        </xs:documentation>
                </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="specializes">
              <xs:annotation>
                      <xs:documentation>
                              A relationshipCode value for indicating that this MPD
is a specialization of the MPD referenced in resourceURI.  This value is the inverse
of generalizes.
                      </xs:documentation>
              </xs:annotation>
            </xs:enumeration>
            <xs:enumeration value="generalizes">
              <xs:annotation>
```

```
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
is a generalization of the MPD referenced in resourceURI.  This value is the inverse
of specializes.
                        </xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="supersedes">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
replaces the MPD referenced in resourceURI.
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="deprecates">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that content in
this MPD is preferred over content in the MPD referenced in resourceURI; and at some
time in the future will supersede the MPD referenced in resourceURI.
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="adapts">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
is an adaptation of the MPD referenced in resourceURI.
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="updates">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
is an incremental update to the resource referenced in resourceURI.  Used by a core
or domain update to identify the domain schema in a NIEM release being incrementally
updated (not replaced).
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="conforms_to">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
conforms to the specification or standard referenced in resourceURI.
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
            </xs:restriction>
        </xs:simpleType>
                <xs:enumeration value="derives_from">
                  <xs:annotation>
                        <xs:documentation>
                                A relationshipCode value for indicating that this MPD
has been derived from another; used to indicate an IEPD is derived from an EIEM (may
have other uses as well).
                        </xs:documentation>
                  </xs:annotation>
                </xs:enumeration>
            </xs:restriction>
        </xs:simpleType>

    <!-- IEP Conformance Targets ================================================= -->
```

```
    <xs:element name="IEPConformanceTarget" type="c:IEPConformanceTargetType">
      <xs:annotation>
              <xs:documentation>
                      A class or category of IEPs which has a set of validity
constraints and a unique identifier. Every IEP is an instance of one or more IEP
Conformance Targets.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="IEPConformanceTargetType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a class or category of IEP, which has a set of
validity constraints and a unique identifier.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:ValidityConstraintWithContext" minOccurs="0"
maxOccurs="unbounded"/>
<!--        <xs:element ref="c:IEPSampleXMLDocument" minOccurs="1"
maxOccurs="unbounded"/> MUST have min one for each IEP CT -->
            <xs:element ref="c:ArtifactOrArtifactSet" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="ValidityConstraintWithContext" abstract="true">
      <xs:annotation>
              <xs:documentation>
                      A data concept for a rule or instructions for validating an IEP
candidate (XML document) using some context within that XML document.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ValidityConstraint" abstract="true"
substitutionGroup="c:ValidityConstraintWithContext">
      <xs:annotation>
              <xs:documentation>
                      A data concept for a rule or instructions for validating an IEP
candidate.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ValidityContext" type="c:ValidityContextType"
substitutionGroup="c:ValidityConstraintWithContext">
      <xs:annotation>
              <xs:documentation>
                      A rule or instructions for validating an IEP candidate within
context defined by an XPath expression.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="ValidityContextType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a rule or instructions for validating an IEP
candidate within context defined by an XPath expression.
```

```
                </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:ValidityConstraint" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="c:xPathText" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>


<!-- Validity Constraints ========================================================= --
>

    <xs:element name="ValidToXPath" type="c:XPathType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that has a an XPath expression that MUST
have an effective Boolean value of "TRUE" when applied to a valid IEP.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="XPathType">
      <xs:annotation>
        <xs:documentation>A data type for an XPath expression.</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="c:xPathText" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:attribute name="xPathText" type="xs:string">
      <xs:annotation>
        <xs:documentation>An XPath expression.</xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:element name="XMLSchemaValid" type="c:XMLSchemaType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that indicates that an artifact must be
locally XML Schema valid against an XML schema described by an XML Catalog file,
starting with a given validation root.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="XMLSchemaType">
      <xs:annotation>
              <xs:documentation>
                      A data type for an XML Schema, indicating an XML Schema against
which an artifact may be validated, or which can be used for other purposes.
c:XMLSchemaDocument identifies the root or starting XML schema document.
              </xs:documentation>
      </xs:annotation>
```

```
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:XMLCatalog"/>
            <xs:element ref="c:XMLSchemaDocument"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="SchematronValid" type="c:SchematronValidationType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that indicates that an artifact must be
valid against the rules carried by a Schematron file, starting with the identified
validation roots.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="SchematronValidationType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a Schematron validation constraint, indicating
a Schematron schema document against which an artifact may be validated as well as a
description of the validation roots for assessment of validity.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:SchematronSchema"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="RelaxNGValid" type="c:RelaxNGValidationType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that indicates that an artifact must be
valid against the rules carried by a RelaxNG schema.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="RelaxNGValidationType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a RelaxNG validation constraint, indicating a
RelaxNG schema document against which an artifact may be validated, as well as a
description of the validation roots for assessment of validity.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:RelaxNGSchema"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
```

```
    </xs:complexType>

    <xs:element name="HasDocumentElement" type="c:QualifiedNamesType"
substitutionGroup="c:ValidityConstraintWithContext">
      <xs:annotation>
              <xs:documentation>
                     A validity constraint that indicates that an artifact has a
document element with a name that is one of the given qualified names.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="QualifiedNamesType">
      <xs:annotation>
        <xs:documentation>A data type for a set of qualified names.
</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="c:qualifiedNameList" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:attribute name="qualifiedNameList" type="c:QualifiedNameListSimpleType">
      <xs:annotation>
        <xs:documentation>A list of qualified names.</xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:simpleType name="QualifiedNameListSimpleType">
      <xs:annotation>
              <xs:documentation>
                      A simple data type denoting a list of qualified names.
              </xs:documentation>
      </xs:annotation>
      <xs:list itemType="xs:QName"/>
    </xs:simpleType>

    <xs:element name="ConformsToConformanceTarget" type="c:ConformanceTargetType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that indicates that an artifact must
conform to the given conformance target.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="ConformanceTargetType">
      <xs:annotation>
              <xs:documentation>
                      A data type for identifying and describing a conformance
target.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="c:conformanceTargetURI" use="required"/>
        </xs:extension>
```

```
      </xs:complexContent>
    </xs:complexType>

    <xs:attribute name="conformanceTargetURI" type="xs:anyURI">
      <xs:annotation>
        <xs:documentation>A URI for a conformance target.</xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:element name="ConformsToRule" type="c:TextRuleType"
substitutionGroup="c:ValidityConstraint">
      <xs:annotation>
              <xs:documentation>
                      A validity constraint that indicates that an artifact must
conform to the given text rule, drafted in a human language.
              </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="TextRuleType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a rule drafted in a human language.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="nc:DescriptionText" minOccurs="0"/>
            <xs:element ref="c:RuleText"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="RuleText" type="nc:TextType">
      <xs:annotation>
        <xs:documentation>A rule written in a human language.</xs:documentation>
      </xs:annotation>
    </xs:element>


<!-- Metadata ========================================================================= --
>

    <xs:element name="MPDInformation" type="c:MPDInformationType">
      <xs:annotation>
        <xs:documentation>A set of descriptive data about an MPD.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:complexType name="MPDInformationType">
      <xs:annotation>
              <xs:documentation>
                      A data type for a set of descriptive data about an MPD.
              </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="structures:ObjectType">
          <xs:sequence>
            <xs:element ref="c:AuthoritativeSource" minOccurs="0"/>
            <xs:element ref="c:CreationDate" minOccurs="0"/>
            <xs:element ref="c:LastRevisionDate" minOccurs="0"/>
            <xs:element ref="c:StatusText" minOccurs="0"/>
            <xs:element ref="c:Relationship" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="c:KeywordText" minOccurs="0" maxOccurs="unbounded"/>
```

```
            <xs:element ref="c:DomainText" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="c:PurposeText" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="c:ExchangePatternText" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element ref="c:ExchangePartnerName" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element ref="c:ExtendedInformation" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="ExtendedInformation" abstract="true">
      <xs:annotation>
            <xs:documentation>
                  A data concept for a user-defined descriptive data about an
MPD.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="AuthoritativeSource" type="nc:EntityType">
      <xs:annotation>
            <xs:documentation>
                  An official sponsoring or authoring organization responsible
for an MPD.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="CreationDate" type="niem-xs:date">
      <xs:annotation>
        <xs:documentation>A date this MPD was published.</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="LastRevisionDate" type="niem-xs:date">
      <xs:annotation>
            <xs:documentation>
                  A date the latest changes to an MPD were published (i.e.,
CreationDate of previous version).
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="StatusText"   type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A description of the current state of this MPD in development;
may also project future plans for the MPD.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="Relationship" type="c:RelationshipType">
      <xs:annotation>
        <xs:documentation>A reference to another MPD related to this MPD.
</xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="KeywordText"  type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A common alias, term, or phrase that would help to facilitate
```

```
search and discovery of this MPD.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="DomainText"   type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A description of the environment or NIEM Domain in which this
MPD is applicable or used.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="PurposeText"  type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A description of the intended usage and reason for which an MPD
exists.
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ExchangePatternText" type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A description of a transactional or design pattern used for
this IEPD (generally, only applicable to IEPDs).
            </xs:documentation>
      </xs:annotation>
    </xs:element>

    <xs:element name="ExchangePartnerName" type="niem-xs:string">
      <xs:annotation>
            <xs:documentation>
                  A name of an entity or organization that uses this MPD
(generally, only applicable to IEPDsP).
            </xs:documentation>
      </xs:annotation>
    </xs:element>

</xs:schema>
```

Appendix B. Example MPD Catalog Document for Cursor on Target

Below is a simple example of an MPD catalog document for a *Cursor on Target* IEPD. The entire IEPD is contained in the **[NIEM MPD Toolkit]**

```
<?xml version="1.0" encoding="US-ASCII"?>
<c:Catalog
    xmlns:cot="http://example.com/cot-niem/0.8/"
    xmlns:c="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"
    xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
    xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/">
    <c:MPD
        c:mpdURI="http://example.com/cot-iepd/0.8beta1/"
        c:mpdClassCode="iepd"
        c:mpdName="cot-iepd"
        c:mpdVersionID="0.8beta1">
        <c:MPDInformation>
            <c:AuthoritativeSource>
                <nc:EntityOrganization>
```

```
                    <nc:OrganizationName>CoT Program Office</nc:OrganizationName>
                    <nc:OrganizationPrimaryContactInformation>

<nc:ContactWebsiteURI>https://partners.mitre.org/sites/CoTUserGroup/</nc:ContactWebsiteURI>

                    </nc:OrganizationPrimaryContactInformation>
                </nc:EntityOrganization>
            </c:AuthoritativeSource>
            <c:CreationDate>2014-03-13</c:CreationDate>
            <c:StatusText>Beta</c:StatusText>
        </c:MPDInformation>
        <c:IEPConformanceTarget structures:id="CoT-NIEM-IEP">
            <nc:DescriptionText>
                An IEP class equivalent to Cursor-on-Target 2.0 messages
            </nc:DescriptionText>
            <c:HasDocumentElement c:qualifiedNameList="cot:Event"/>
            <c:XMLSchemaValid>
                <c:XMLCatalog c:pathURI="xml-catalog.xml"/>
                <c:XMLSchemaDocument c:pathURI="extension/cot-neim.xsd"/>
            </c:XMLSchemaValid>
            <c:SchematronValid>
                <c:SchematronSchema c:pathURI="schematron/business-rules.sch"/>
            </c:SchematronValid>
       <!-- <c:EXISchema>
                <c:XMLCatalog c:pathURI="xml-catalog-exi.xml"/>
                <c:XMLSchemaDocument c:pathURI="extension/cot-niem.xsd"/>
            </c:EXISchema> -->
            <c:IEPSampleXMLDocument c:pathURI="iep-samples/cot-iep1.xml"/>
        </c:IEPConformanceTarget>
        <c:ReadMe c:pathURI="read-me.txt"/>
        <c:MPDChangeLog c:pathURI="changelog.txt"></c:MPDChangeLog>
        <c:Wantlist c:pathURI="niem/wantlist.xml"/>
        <c:ExtensionSchemaDocument c:pathURI="extension/cot-niem.xsd"/>
        <c:ExtensionSchemaDocument c:pathURI="extension/cot-niem-codes.xsd"/>
        <c:ReferenceSchemaDocument c:pathURI="extension/milops-future.xsd"/>
    </c:MPD>
</c:Catalog>
```

Appendix C. Schematron Rules for MPDs

The following Schematron rule files represent a number of the rules defined in this MPD Specification. None of these rules have been tested. For now they should be considered experimental (alpha). In the future, more Schematron rules will be designed, tested, and included with this specification and **[NIEM MPD Toolkit]**. This is not likely to occur until after this specification has been approved and released.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   queryBinding="xslt2"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://purl.oclc.org/dsdl/schematron">

<title>Rules about MPD catalog artifacts</title>

<ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>
<ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
<ns prefix="c" uri="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"/>
<ns prefix="structures" uri="http://release.niem.gov/niem/structures/3.0/"/>

<pattern>
  <rule context="c:IEPConformanceTarget">
    <assert test="exists(@structures:id)"
      >Rule 4-21: An element c:IEPConformanceTarget MUST have an attribute
```

```
structures:id.</assert>
</pattern>

</schema>
<!--
  Local Variables:
  mode: sgml
  indent-tabs-mode: nil
  fill-column: 9999
  End:
-->
<?xml version="1.0" encoding="UTF-8"?>
<schema
   queryBinding="xslt2"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://purl.oclc.org/dsdl/schematron">
   xmlns:lf="http://niem.gtri.gatech.edu/local-functions/2014-04-30-1349">

<title>Rules about an MPD catalog document in a well-formed MPD</title>

<ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>
<ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
<ns prefix="c" uri="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"/>
<ns prefix="structures" uri="http://release.niem.gov/niem/structures/3.0/"/>
<ns prefix="xmlcatalog" uri="urn:oasis:names:tc:entity:xmlns:xml:catalog"/>>
<ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron"/>

<xsl:function name="lf:has-effective-conformance-target-identifier" as="xs:boolean">
  <xsl:param name="context" as="element()"/>
  <xsl:param name="match" as="xs:anyURI"/>
  <xsl:variable name="effective-conformance-targets-attribute"
            as="attribute()?"
            select="(root($context)//*[exists(@ct:conformanceTargets)])
[1]/@ct:conformanceTargets"/>
  <xsl:sequence select="if (empty($effective-conformance-targets-attribute))
                    then false()
                    else some $effective-conformance-target-string
                         in tokenize(normalize-space(string($effective-conformance-
targets-attribute)), ' ')
                         satisfies (
                           $effective-conformance-target-string castable as xs:anyURI
                           and xs:anyURI($effective-conformance-target-string) =
$match)"/>
</xsl:function>


<pattern>
  <rule context="c:*[exists(@c:pathURI)]">
    <assert test="unparsed-text-available(resolve-uri(@c:pathURI, base-uri(.)))"
      >Rule 5-2: The value of a c:pathURI attribute MUST resolve to a resource.
</assert>
</pattern>

<pattern>
  <rule context="c:XMLCatalog[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                    doc-available($uri)
                    and doc($uri)/xmlcatalog:catalog)"
      >Rule 5-3: The value of a c:pathURI attribute owned by an element c:XMLCatalog
MUST resolve to an XML catalog document.</assert>
</pattern>

<pattern>
  <rule context="c:IEPSampleXMLDocument[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies
                    doc-available($uri)"
```

```
      >Rule 5-6: The value of a c:pathURI attribute owned by an element
c:IEPSampleXMLDocument MUST resolve to an XML document.</assert>
</pattern>

<pattern>
  <rule context="c:ExternalSchemaDocument[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                  doc-available($uri)
                  and doc($uri)/xs:schema)"
      >Rule 5-8: The value of a c:pathURI attribute owned by an element
c:ExternalSchemaDocument MUST resolve to an XML schema document.</assert>
</pattern>

<pattern>
  <rule context="c:ReferenceSchemaDocument[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                  doc-available($uri)
                  and doc($uri)/xs:schema
                  and lf:has-effective-conformance-target-
identifier(doc($uri)/xs:schema,
                           'http://reference.niem.gov/niem/specification/naming-and-
design-rules/3.0/#ReferenceSchemaDocument'))"
      >Rule 5-9: The value of a c:pathURI attribute owned by an element
c:ReferenceSchemaDocument MUST resolve to a NIEM reference schema document.</assert>
</pattern>

<pattern>
  <rule context="c:ExtensionSchemaDocument[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                  doc-available($uri)
                  and doc($uri)/xs:schema
                  and lf:has-effective-conformance-target-
identifier(doc($uri)/xs:schema,
                           'http://reference.niem.gov/niem/specification/naming-and-
design-rules/3.0/#ExtensionSchemaDocument'))"
      >Rule 5-10: The value of a c:pathURI attribute owned by an element
c:ExtensionSchemaDocument MUST resolve to a NIEM extension schema document.</assert>
</pattern>

<pattern>
  <rule context="c:SubsetSchemaDocument[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                  doc-available($uri)
                  and doc($uri)/xs:schema)"
      >Rule 5-11: The value of a c:pathURI attribute owned by an element
c:SubsetSchemaDocument MUST resolve to an XML schema document.</assert>
</pattern>

<pattern>
  <rule context="c:SchematronSchema[exists(@c:pathURI)]">
    <assert test="some $uri in resolve-uri(@c:pathURI, base-uri(.)) satisfies (
                  doc-available($uri)
                  and doc($uri)/sch:schema)"
      >Rule 5-13: The value of a c:pathURI attribute owned by an element
c:SchematronSchema MUST resolve to a Schematron schema.</assert>
</pattern>

</schema>
<!--
  Local Variables:
  mode: sgml
  indent-tabs-mode: nil
  fill-column: 9999
  End:
-->
<?xml version="1.0" encoding="UTF-8"?>
```

```
<schema
   queryBinding="xslt2"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://purl.oclc.org/dsdl/schematron">
   xmlns:lf="http://niem.gtri.gatech.edu/local-functions/2014-04-30-1349">

<title>Rules about an XML catalog document in a well-formed MPD</title>

<ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>
<ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
<ns prefix="c" uri="http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"/>
<ns prefix="structures" uri="http://release.niem.gov/niem/structures/3.0/"/>
<ns prefix="xmlcatalog" uri="urn:oasis:names:tc:entity:xmlns:xml:catalog"/>>
<ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron"/>

</schema>
<!--
  Local Variables:
  mode: sgml
  indent-tabs-mode: nil
  fill-column: 9999
  End:
-->
```

Appendix D. Common MPD Artifacts

Notes:

- (R) in artifact name indicates a required artifact for every MPD.
- (ref) in definition indicates name is hotlinked to a definition in specification text.
- * in filename syntax indicates wildcard.

Artifact name	Filename syntax	Definition
(R) ·MPD catalog document·	mpd-catalog.xml	(ref)
(R) ·read-me artifact·	read-me.*	(ref)
(R) ·change log·	changelog.*	(ref)
(R) ·conformance assertion·	conformance-assertion.*	(ref)
·XML catalog document·	xml-catalog.xml	(ref)
conformance report	conformance-report.*	a formal report on conformance generated by a NIEM-aware tool
MPD catalog extension ·XML catalog document·	mpd-catalog-extension-xml-catalog.xml	an XML catalog that identifies an MPD catalog extension schema document (ref)
MPD catalog ·extension schema document·	mpd-catalog-extension.xsd	a XML schema document that extends an MPD catalog schema (ref)
·subset schema document·	*.xsd	(ref)
·NIEM wantlist·	wantlist.xml	(ref)
·extension schema document·	*.xsd	(ref)

·external schema document·	*.xsd	(ref)
·reference schema document·	*.xsd	(ref)
IEP sample ·XML document·	*.xml	an XML document instance that exemplifies an ·IEP conformance target· defined by a `c:IEPConformanceTarget` (ref)
Schematron schema document	*.sch	a business rule schema in ISO Schematron format (**[ISO Schematron]**)
RelaxNG schema document	*.rng	a business rule schema in ISO RelaxNG format (**[ISO RelaxNG]**)
documentation	*.*	a textual or graphic artifact containing notes, instructions, guidance, etc.
application information	*.*	a tool-specific artifact used, generated, exported, imported, etc. by a specific tool; includes models, databases, configuration files, graphics, etc.

Appendix E. Conformance Assertion Example

A NIEM conformance assertion is a required artifact for an IEPD or EIEM. The following is a simple example of a conformance assertion (in this case, a self assertion by the author, but with a little assistance from colleagues. The concept is to provide an implementer with some information that indicates how well an IEPD or EIEM has been checked for quality and conformance with respect to XML Schema and NIEM. The assertion can be as simple as the *assertion* clause. However, clearly the more detail that is provided, the stronger the case for conformance and quality will be.

NIEM Conformance Assertion

Assertion: This NIEM IEPD is certified to conform to the following NIEM specifications:

- Conformance v3.0, URI = http://reference.niem.gov/niem/conformance/3.0/
- Naming and Design Rules v3.0, URI = http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/
- Model Package Description v3.0, URI = http://reference.niem.gov/niem/specification/model-package-description/3.0/

Authored by: John Doe, IEPD developer, ABC Company, Inc.

Certified by: John Doe

Certification Date: 1 May 2014

Details of conformance verification

1. How NIEM conformance was verified:
    a. Automatic (tool) checks were performed with NIEM-aware and XML Schema tools identified below.
    b. Manual (subjective) checks on conformance rules were performed by the author/certifier.
    c. A general manual (subjective) cross-check for adherence to conformance rules and quality assurance was performed by several of the author's colleagues who understand NIEM and XML Schema.
    d. Business requirements associated with the information exchange defined by this IEPD were verified by two subject matter experts who are colleagues of the author.

2. Tools employed:
    a. NIEM Schema Subset Generator (SSGT) (for NIEM 3.0) generated all subset schema documents.
    b. XMLSpy Professional 2014r2 SP1 was used for initial editing and validation of extension schema documents, constraint schema documents, and all other XML artifacts. This tool also validated a number of NIEM NDR rules.
    c. oXygen XML Editor v15.2 was used to cross-validate XML schemas and XML instances.
    d. Xerces 2.7.0-0 was used for cross-validation of XML schemas and XML instances.
    e. NIEM ConTesA (Conformance Testing Assistant) was used for NIEM NDR and MPD validation.

3. Results:
    a. No known major issues remain.
    b. All XML artifacts are well-formed and valid.
    c. Attached conformance report from NIEM ConTesA indicates 100% pass for all automatically checked NIEM rules.
    d. Author and SMEs verified that several warnings by tools are not relevant to this IEPD.

Appendix F. Guidance for IEPD Directories (non-normative)

NIEM releases, core updates, and domain updates generally follow a consistent directory organization. When employing release and updates within IEPDs and EIEMs whether as-is or as subsets, users are encouraged to maintain their original directory structures. However, aside from applicable rules previously stated in the preceding sections, there are no normative rules for organizing directories within IEPDs or EIEMs.

Guidance for directory structuring may be useful to authors, especially in the case of a relatively simple IEPD or EIEM with a single schema document subset, and a few extension and external schema documents. The following are common non-normative practice for IEPD directories:

1. Create a root directory for the IEPD from the name and version identifier of the IEPD. For example `my_iepd-3.2rev4`. (Rule 7-3, above)
2. The following artifacts are required to be in the ·MPD root directory· (Note: `.*` indicates any format):
   - `mpd-catalog.xml` (Rule 4-1, above)
   - `changelog.*` (Rule 4-17, above)
   - `read-me.*` (Rule 4-18, above)
   - `conformance-assertion.* (Rule 4-24, above)`
3. If extending the ·MPD catalog document·, then per Rule 4-4, above `mpd-catalog-extension-xml-catalog.xml` must reside in the same relative directory as the `mpd-catalog.xml` it supports (normally, the ·MPD root directory·). `mpd-catalog-extension.xsd` can be anywhere in the MPD because `mpd-catalog-extension-xml-catalog.xml` must correlate its namespace to its URI. However, we recommend both artifacts be co-located in the MPD root directory for visibility:
   - `mpd-catalog-extension.xsd`
   - `mpd-catalog-extension-xml-catalog.xml` (Rule 4-4, above)
4. Create the following directories within the ·MPD root directory·:
   - `base-xsd` — will contain the NIEM subset and its associated extension, external, and custom NIEM schema documents. These are the NIEM XML schema documents used to validate conformance of an instance XML document. Subdirectories under `base-xsd` may include:
     - `niem` — a NIEM schema subset organized exactly as the SSGT generates it (including its `wantlist.xml` and `xml-catalog.xml`).
     - `extension` — for NIEM extension schema documents.
     - `external` — for non-NIEM standards used by the IEPD.
     - `niem-custom` — for NIEM schema documents that may be customized (extended or restricted) such as `structures.xsd`.
   - `constraint-xsd` — will contain constraint schema documents organized as necessary. Usually this schema document set will be organized similarly to schema documents in `base-xsd` because it is customary to start with the base schema set and constrain it as necessary.
5. If using Schematron, create a `schematron` subdirectory for any Schematron schemas (or create the appropriate subdirectory name for any other kinds of business rule artifacts).
6. If using EXI schema documents, create a `exi-xsd` subdirectory for these.
7. Create an `iep-sample` subdirectory for sample IEPs (Rule 4-23, above).
8. If more documentation artifacts (e.g., text, graphics, media) are necessary, create a `documentation` subdirectory for miscellaneous explanatory documentation. As needed, create additional subdirectories within this one to organize documentation artifacts. The `read-me.*` artifact in the ·MPD root directory· should refer to or index documentation in this subdirectory.
9. If necessary, create an `application-info` subdirectory for tool-specific artifacts (inputs, outputs, imports, exports, models, etc.). Again, as needed, use additional subdirectories to organize artifacts of this nature. `read-me.*` can and should also refer to or index artifacts in this subdirectory.
10. Maintain a ·NIEM wantlist· within the same subdirectory as the subset it generates (Rule 6-1, above).
11. Maintain an `xml-catalog.xml` closest to the XML schema document set it relates to. Use `nextCatalog` elements as needed to help maintain this proximity (Rule 4-20, above).
12. Finally, if it becomes necessary to maintain multiple `constraint-xsd` or `extension` subdirectories (or any other subdirectories) together in the same directory, then simply suffix each directory name with a distinct character string (for example, `extension1`, `extension2`, etc.; or `extension-abc`,

`extension-zyx`, etc.)

Obviously, there are many other ways to organize for more complex business requirements in which a single IEPD or EIEM employs multiple releases, subsets, constraint sets, core updates, and domain updates. Regardless of directory organization and file naming, an IEPD or EIEM author must always configure all IEP conformance targets using the MPD catalog `c:IEPConformanceTarget` element and the appropriate validation artifacts (such as XML catalogs, Schematron schemas, RelaxNG schemas, etc.).

The guidance above results in an IEPD directory structure that appears below. Notes are in parentheses. Filenames within the `extension`, `external`, `schematron`, and `iep-sample` subdirectories are non-normative examples. Authors are free to assign names for such files according to their own requirements (if they do not violate rules in this specification).

```
/my_iepd-3.2rev4        (root directory of IEPD archive)

        mpd-catalog.xml
        mpd-catalog-extension.xsd
        mpd-catalog-extension-xml-catalog.xml
        changelog.*
        conformance-assertion.*
        read-me.*

        /base-xsd
                /niem                   (subset)
                        /adapters
                        /appinfo
                        /codes
                        /conformanceTargets
                        /domains
                        /external
                        /localTerminology
                        /niem-core
                        /proxy
                        /structures
                        wantlist.xml
                        xml-catalog.xml

                /niem-custom            (extension/restriction of structures)
                        structures.xsd

                /extension
                        query.xsd
                        response.xsd
                        extension1.xsd
                        extension2.xsd
                        ...
                        xml-catalog.xml

                /external
                        /stix
                        /ic-ism
                        ...
                        xml-catalog.xml

        /constraint-xsd                 (constraint schema documents)
                /niem                   (constraints on subset)
                        /adapters
                        /appinfo
                        /codes
                        /conformanceTargets
                        ...
                        wantlist.xml
                        xml-catalog.xml
```

```
        /extension              (constraints on extensions)
                query.xsd
                extension1.xsd
                xml-catalog.xml

/exi-xsd
        gml.xsd
        xs.xsd
        ...

/schematron
        business-rules1.sch
        business-rules2.sch
        ...

/iep-sample
        query.xml
        request.xml
        ...

/application-info
        ... (tool inputs, outputs, etc.)

/documentation
        ... (human readable documentation)
```

Appendix G. Acronyms and Abbreviations

Acronym / Abbreviation	Literal or Definition
ASCII	American Standard Code for Information Interchange
BIEC	Business Information Exchange Component
CSV	Comma Separated Value (file format)
CU	Core update
CUI	Controlled Unclassified Information
DU	Domain update
EBV	Effective Boolean Value
EIEM	Enterprise Information Exchange Model
GIF	Graphic Interchange Format
GML	Geospatial Markup Language
HTML	Hyper Text Markup Language
IEP	Information Exchange Package
IEPD	Information Exchange Package Documentation
JPG	Joint Photographic (Experts) Group
LEXS	Logical Entity Exchage Specifications
MPD	Model Package Description
NDR	Naming and Design Rules
NIEM	National Information Exchange Model
NTAC	NIEM Technical Architecture Committee
PDF	Portable Document Format
PMO	Program Management Office
PNG	Portable Network Graphic

QName	Qualified Name (XML Schema: qualified by a namespace prefix)
RAR	Roshal Archive; a compressed archive file format named for its developer, Eugene Roshal
Rel	Release (NIEM)
SSGT	Schema Subset Generation Tool
UML	Unified Modeling Language
UPA	Unique Particle Attribution
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XS	XML Schema (namespace prefix)
XSD	XML Schema Definition
XSI	XML Schema Instance (namespace prefix)
XSLT	Extensible Stylesheet Language Transformation

Appendix H. References

[FEA Data Reference Model]: *The Federal Enterprise Architecture Data Reference Model*, Version 1.0, September 2004. Available from `http://xml.gov/documents/completed/DRMv1.pdf`. A more recent DRM Version 2.0, 17 November 2005 is available from `http://www.whitehouse.gov/omb/assets/egov_docs/DRM_2_0_Final.pdf`

[GJXDM IEPD Guidelines]: *GJXDM Information Exchange Package Documentation Guidelines*, Version 1.1, Global XML Structure Task Force (GXSTF), 2 March 2005. Available from `http://it.ojp.gov/documents/global_jxdm_IEPD_guidelines_v1_1.pdf`

[ISO 11179-4]: *ISO/IEC 11179-4 Information Technology — Metadata Registries (MDR) — Part 4: Formulation of Data Definitions Second Edition,* 15 July 2004. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c035346_ISO_IEC_11179-4_2004(E).zip`.

[ISO 11179-5]: *ISO/IEC 11179-5:2005, Information technology — Metadata registries (MDR) — Part 5: Naming and identification principles*. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip`.

[ISO RelaxNG]: *Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*, ISO/IEC 19757-2:2008, Second Edition, 15 December 2008. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip`. See also `http://relaxng.org`.

[ISO Schematron]: *Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron*, ISO/IEC 19757-3:2006(E), First Edition, 1 June 2006. Available from `http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip`.

[Logical Entity Exchange Specifications]: *Logical Entity Exchange Specifications*, Version 4.0, 27 July

2011. Available from http://130.207.211.107/content/downloads.

[NIEM BIEC]: *Business Information Exchange Components (BIEC)*, Version 1.0, NIEM Technical Architecture Committee (NTAC), March 2011. Available from http://reference.niem.gov/niem/guidance/business-information-exchange-components/1.0/.

[NIEM Conformance]: *NIEM Conformance*, Version 3.0, NIEM Technical Architecture Committee (NTAC), [day month TBD] 2014. Available from http://reference.niem.gov/niem/specification/conformance/3.0/.

[NIEM Conformance Target Attribute Specification]: *NIEM Conformance Target Attribute Specification*, Version 1.0, NIEM Technical Architecture Committee (NTAC), [day month TBD] 2014. Available from http://reference.niem.gov/niem/specification/conformance-target-attribute/1.0/.

[NIEM Concept of Operations]: *NIEM Concept of Operations*, Version 0.5, NIEM Program Management Office, 9 January 2007. Available from http://reference.niem.gov/niem/guidance/concept-of-operations/.

[NIEM Domain Update Specification]: *NIEM Domain Update Specification*, Version 1.0, NIEM Technical Architecture Committee (NTAC), 5 November 2010. Available from http://reference.niem.gov/niem/specification/domain-update/1.0/.

[NIEM High-Level Tool Architecture]: *NIEM High-Level Tool Architecture*, Version 1.1, NIEM Technical Architecture Committee, 1 December 2008. Available from http://reference.niem.gov/niem/specification/high-level-tool-architecture/1.1/.

[NIEM High-Level Version Architecture]: *NIEM High Level Version Architecture (HLVA)*, Version 1.0, NIEM Technical Architecture Committee, 2008. Available from http://reference.niem.gov/niem/specification/high-level-version-architecture/1.0/.

[NIEM IEPD Requirements]: *Requirements for a National Information Exchange Model (NIEM) Information Exchange Package Documentation (IEPD) Specification*, Version 2.1, June 2006. Available from http://reference.niem.gov/niem/guidance/iepd-requirements/2.1/.

[NIEM Implementation Guidance]: "NIEM Implementation Guide", NIEM Program Management Office. Available from https://www.niem.gov/aboutniem/grant-funding/Pages/implementation-guide.aspx.

[NIEM Introduction]: *Introduction to the National Information Exchange Model (NIEM)*, Version 0.3, NIEM Program Management Office, 12 February 2007. Available from http://reference.niem.gov/niem/guidance/introduction/.

[NIEM MPD Specification]: *NIEM Model Package Description (MPD) Specification*, Version 3.0, NIEM Technical Architecture Committee (NTAC), [day month TBD] 2014. Available from http://reference.niem.gov/niem/specification/model-package-description/3.0/.

[NIEM MPD Toolkit]: NIEM Model Package Description Toolkit, Version 3.0, NIEM Technical Architecture Committee (NTAC), [day month TBD] 2014. Available from http://referencce.niem.gov/niem/resource/mpd/3.0/.

[NIEM NDR]: *NIEM Naming and Design Rules (NDR)*, Version 3.0, NIEM Technical Architecture Committee (NTAC), [day month TBD] 2014. Available from http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/.

[NIEM SSGT]: NIEM Schema Subset Generation Tool (SSGT). Available from http://tools.niem.gov/niemtools/ssgt/index.iepd.

[NIEM User Guide]: *NIEM User Guide*, Volume 1, U.S. Department of Justice, Office of Justice Programs, (date unknown). Available from http://reference.niem.gov/niem/guidance/user-guide/vol1/.

[XML Catalogs]: *XML Catalogs*, Organization for the Advancement of Structured Information Standards (OASIS) Standard v1.1, 7 October 2005. Available from https://www.oasis-open.org/committees/download.php/14809/std-entity-xml-catalogs-1.1.html.

[Principles of Data Integration]: Doan, A., Halevy, A., and Ives, X. (2012), *Principles of Data Integration*, New York, NY: Morgan Kaufmann.

[RAR]: RARLAB WinRAR, http://win-rar.com

[RFC 2119 Key Words]: Bradner, S., *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. Available from http://www.ietf.org/rfc/rfc2119.txt.

[RFC 2141 URN Syntax]: Moats, R., *URN Syntax*, IETF RFC 2141, May 1997. Available from http://tools.ietf.org/html/rfc2141.

[RFC 3986 URI]: Berners-Lee, T., et al., *Uniform Resource Identifier (URI): Generic Syntax*, Request for Comments 3986, Network Working Group, January 2005. Available from http://tools.ietf.org/html/rfc3986.

[W3-EXI]: *Efficient XML Interchange (EXI) Format*, Version 1.0, W3C Recommendation, 10 March 2011. Available from http://www.w3.org/TR/2011/REC-exi-20110310/.

[W3-XML]: *Extensible Markup Language (XML)*, Version 1.0, Fifth Edition, W3C Recommendation 26 November 2008. Available from http://www.w3.org/TR/2008/REC-xml-20081126/.

[W3-XML-InfoSet]: *XML Information Set*, Second Edition, W3C Recommendation 4 February 2004. Available from http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[W3-XML-Namespaces]: *Namespaces in XML*, Second Edition, World Wide Web Consortium 16 August 2006. Available from http://www.w3.org/TR/2006/REC-xml-names-20060816/.

[W3C XML Schema Datatypes]: *XML Schema Part 2: Datatypes*, Second Edition, W3C Recommendation 28 October 2004. Available from http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[W3C XML Schema Structures]: *XML Schema Part 1: Structures*, Second Edition, W3C Recommendation 28 October 2004. Available from http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[W3C XPath 2.0]: *XML Path Language (XPath) 2.0*, Second Edition, W3C Recommendation 14 December 2010. Available from http://www.w3.org/TR/2010/REC-xpath20-20101214/.

[XSLT v1.0]: *XSL Transformations (XSLT)*, Version 1.0, W3C Recommendation 16 November 1999. Available from http://www.w3.org/TR/1999/REC-xslt-19991116.

[XSLT v2.0]: *XSL Transformations (XSLT)*, Version 2.0, W3C Recommendation 23 January 2007. Available from http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[PKZIP]: *APPNOTE.TXT - .ZIP File Format Specification*, Version: 6.3.2, Revised: 28 September 2007, Copyright (c) 1989 - 2007 PKWare Inc. Available from http://www.pkware.com/documents/casestudies/APPNOTE.TXT.

Appendix I. Index of Definitions

- well-formed MPD: Section 5.1, *Well Formed MPD (WF-MPD)*
- XML catalog document: Section 4.5, *XML Catalogs*
- XML document: Section 2.6, *XML Validation*
- XML Schema: Section 2.6, *XML Validation*
- XML schema assembly: Section 2.6, *XML Validation*
- XML schema document: Section 2.6, *XML Validation*
- XML schema validation: Section 2.6, *XML Validation*

Appendix J. Index of Rules

- Rule 3-1: Section 3.2.1, *Basic Subset Concepts*
- Rule 3-2: Section 3.2.3, *Subset Schema Document Namespaces*
- Rule 3-3: Section 3.5, *Constraint Schema Documents and Document Sets*
- Rule 4-1: Section 4.1, *NIEM MPD Catalog*
- Rule 4-2: Section 4.1, *NIEM MPD Catalog*
- Rule 4-3: Section 4.1, *NIEM MPD Catalog*
- Rule 4-4: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-5: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-6: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-7: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-8: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-9: Section 4.1.3, *Extending an MPD Catalog*
- Rule 4-10: Section 4.2.1, *Version Numbering Scheme*
- Rule 4-11: Section 4.2.2, *URI Scheme for MPDs*
- Rule 4-12: Section 4.2.3, *URI Scheme for MPD Artifacts*
- Rule 4-13: Section 4.2.3, *URI Scheme for MPD Artifacts*
- Rule 4-14: Section 4.2.3, *URI Scheme for MPD Artifacts*
- Rule 4-15: Section 4.3.1, *Change Log for Releases and Core/Domain Updates*
- Rule 4-16: Section 4.3.1, *Change Log for Releases and Core/Domain Updates*
- Rule 4-17: Section 4.3.2, *Change Log for IEPDs and EIEMs*
- Rule 4-18: Section 4.4, *Read-Me Artifact*
- Rule 4-19: Section 4.4.1, *Read-me Content*
- Rule 4-20: Section 4.5, *XML Catalogs*
- Rule 4-21: Section 4.6, *Information Exchange Packages*
- Rule 4-22: Section 4.6.2, *Declaring Validity Constraints*
- Rule 4-23: Section 4.6.3, *IEP Sample XML Instance Documents*
- Rule 4-24: Section 4.7, *Conformance Assertion*
- Rule 4-25: Section 4.7, *Conformance Assertion*
- Rule 5-1: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-2: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-3: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-4: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-5: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-6: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-7: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-8: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-9: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-10: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-11: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-12: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-13: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-14: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-15: Section 5.1, *Well Formed MPD (WF-MPD)*

- Rule 5-16: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-17: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-18: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-19: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-20: Section 5.1, *Well Formed MPD (WF-MPD)*
- Rule 5-21: Section 5.2, *IEP*
- Rule 5-22: Section 5.2, *IEP*
- Rule 5-23: Section 5.2, *IEP*
- Rule 5-24: Section 5.3, *Full NIEM IEP (FN-IEP)*
- Rule 5-25: Section 5.3, *Full NIEM IEP (FN-IEP)*
- Rule 5-26: Section 5.3, *Full NIEM IEP (FN-IEP)*
- Rule 6-1: Section 6.1, *NIEM Wantlist*
- Rule 7-1: Section 7, *Organization, Packaging, and Other Criteria*
- Rule 7-2: Section 7, *Organization, Packaging, and Other Criteria*
- Rule 7-3: Section 7, *Organization, Packaging, and Other Criteria*
- Rule 7-4: Section 7.2, *MPD File Name Syntax*
- Rule 7-5: Section 7.2, *MPD File Name Syntax*
- Rule 7-6: Section 7.2, *MPD File Name Syntax*
- Rule 7-7: Section 7.3, *Artifact Links to Other Resources*
- Rule 7-8: Section 7.3, *Artifact Links to Other Resources*
- Rule 7-9: Section 7.4, *IEPD Completeness*
- Rule 7-10: Section 7.4, *IEPD Completeness*
- Rule 7-11: Section 7.4, *IEPD Completeness*